

# MacTech®

*The Journal of Macintosh Technology and Development*

## Interactive Development with ActiveDeveloper

*An introduction to Interactive Development and a review of Inter\*ACTIVE*

*By Aidan Reel*



Copyright 1994-2003

Inter\*ACTIVE Technology

All rights reserved

### **Also Inside:**

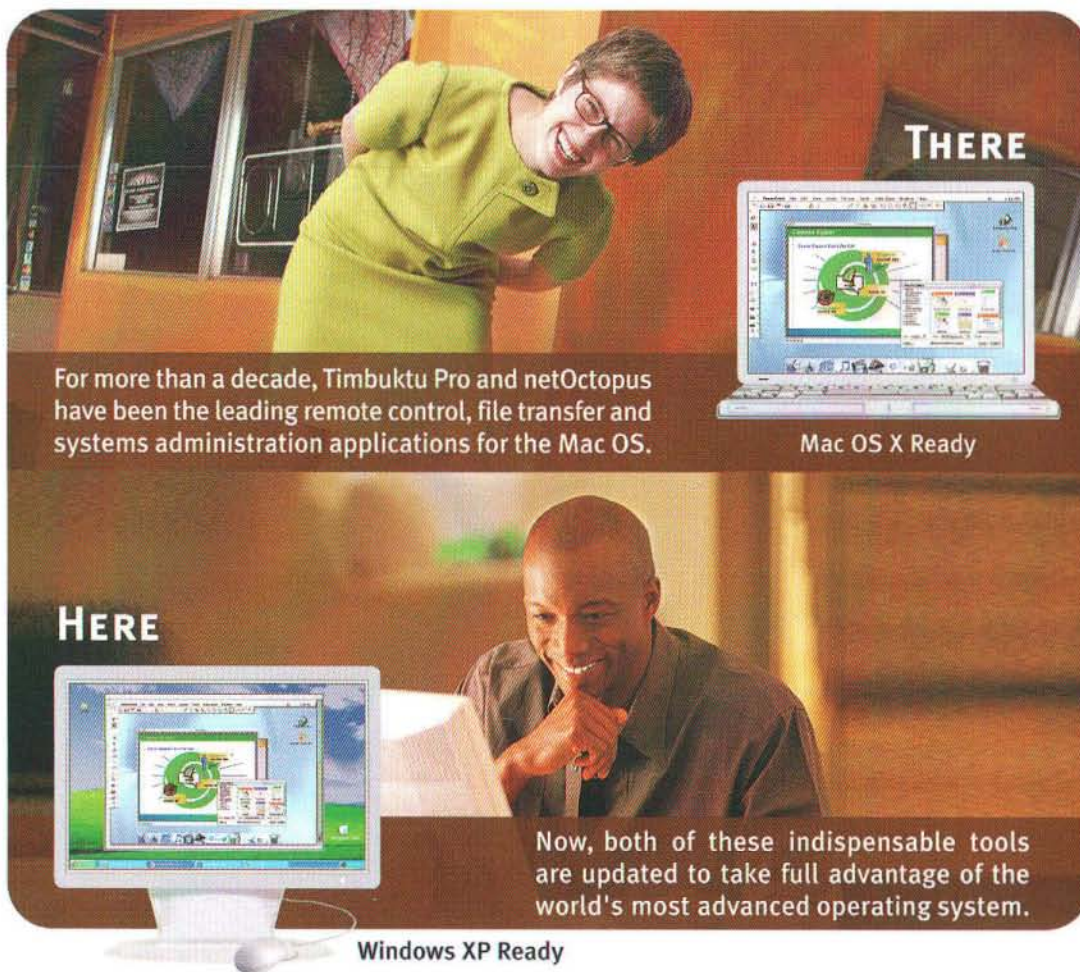
**Notes from the REALWorld Conference**  
**Object-Oriented Programming with REALbasic**  
**A Look at Panther Server**  
**The NEW Script Editor**  
**Database Modification with a GUI**

\$8.95 US  
\$12.95 Canada  
ISSN 1067-8360  
Printed in U.S.A.





# Stay In Control Wherever You Go.



**THERE**

For more than a decade, Timbuktu Pro and netOctopus have been the leading remote control, file transfer and systems administration applications for the Mac OS.

Mac OS X Ready

**HERE**

Now, both of these indispensable tools are updated to take full advantage of the world's most advanced operating system.

Windows XP Ready

## Timbuktu Pro

Whether you're at home or at work, Timbuktu Pro allows you to operate distant computers as if you were sitting in front of them, transfer files or folders quickly and easily, and communicate by instant message, text chat, or voice intercom.

<http://www.timbuktopro.com>

## netOctopus

Intuitive and powerful, netOctopus can manage a network of ten or 10,000 computers. Inventory computers, software and devices on your network; distribute software; configure remote computers; and create custom reports on the fly.

<http://www.netoctopus.com>

Learn more, try it, or buy it online. Call us at **1-800-485-5741**.



timbuktu® • netOctopus®

netopia®



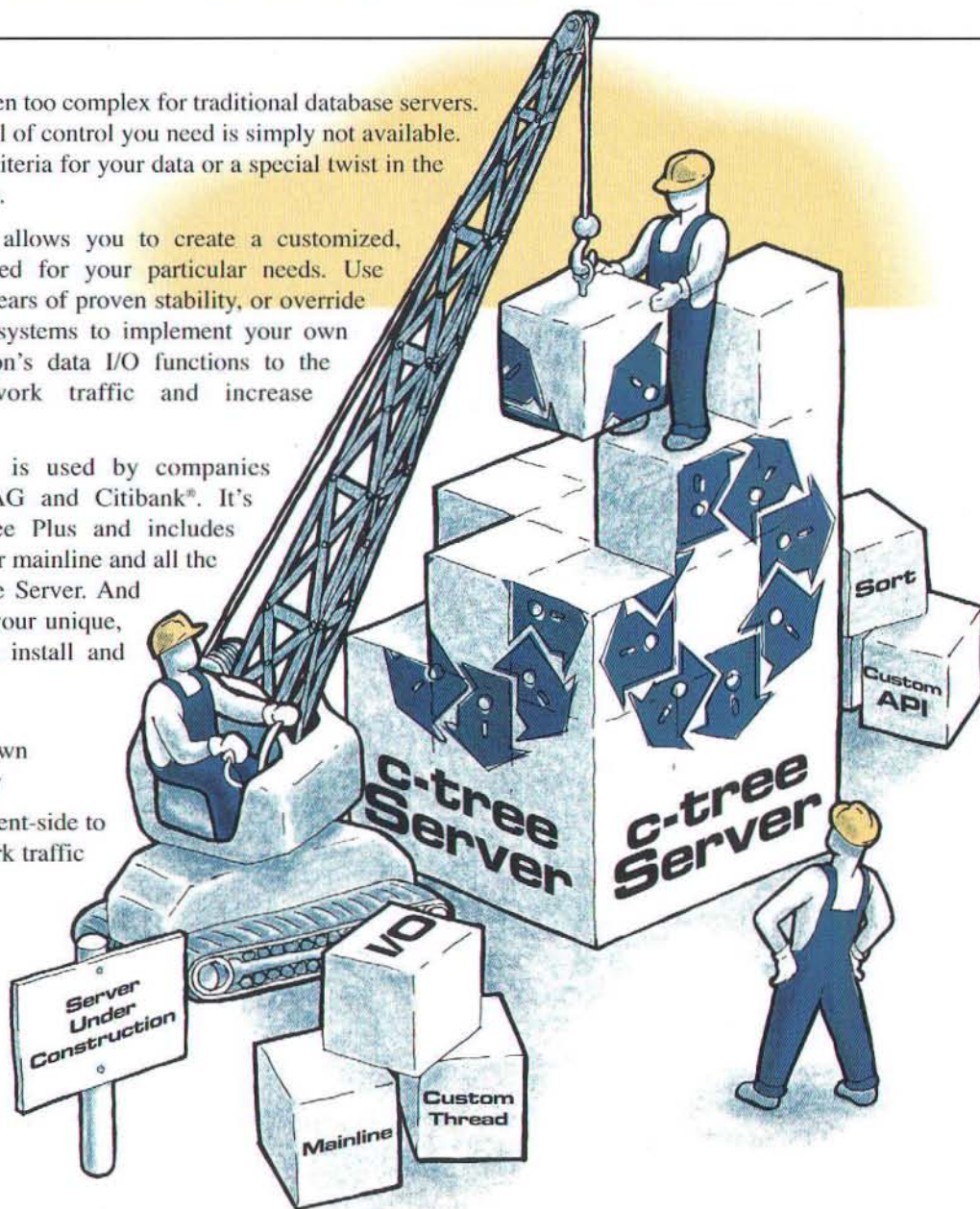
# CUSTOMIZE YOUR DATABASE SERVER WITH THE C-TREE<sup>®</sup> SERVER SDK

Today's database demands are often too complex for traditional database servers. The functionality and precise level of control you need is simply not available. Perhaps you need alternate sort criteria for your data or a special twist in the threading or communication logic.

FairCom's c-tree<sup>®</sup> Server SDK allows you to create a customized, industrial-strength server designed for your particular needs. Use FairCom's kernel, with over 20 years of proven stability, or override functionality within specific subsystems to implement your own subtleties. Move your application's data I/O functions to the server-side to decrease network traffic and increase performance!

FairCom's c-tree Server SDK is used by companies worldwide such as Software AG and Citibank<sup>®</sup>. It's integrated seamlessly into c-tree Plus and includes complete source code to the server mainline and all the interface subsystems to the c-tree Server. And best of all, once you've created your unique, customized server, it is easy to install and administer: no DBA required!

- Enhance our server with your own custom server-side functionality
- Move functionality from the client-side to the server-side to reduce network traffic and increase performance
- Modify or replace entire server subsystems
- Complete source for the server mainline, key server subsystems, and client-side
- Flexible OEM licensing



Visit [www.faircom.com/ep/mt/sdk](http://www.faircom.com/ep/mt/sdk) today to take control of your server!



**FairCom<sup>®</sup>**  
[www.faircom.com](http://www.faircom.com)

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

DBMS Since 1979 • 800.234.8180 • [info@faircom.com](mailto:info@faircom.com)

Other company and product names are registered trademarks or trademarks of their respective owners.

© 2002 FairCom Corporation



## How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail**?

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

### DEPARTMENTS

**Orders, Circulation, & Customer Service**

**Press Releases**

**Ad Sales**

**Editorial**

**Programmer's Challenge**

**Online Support**

**Accounting**

**Marketing**

**General**

**Web Site (articles, info, URLs and more...)**

### E-Mail/URL

[cust\\_service@mactech.com](mailto:cust_service@mactech.com)

[press\\_releases@mactech.com](mailto:press_releases@mactech.com)

[ad\\_sales@mactech.com](mailto:ad_sales@mactech.com)

[editorial@mactech.com](mailto:editorial@mactech.com)

[prog\\_challenge@mactech.com](mailto:prog_challenge@mactech.com)

[online@mactech.com](mailto:online@mactech.com)

[accounting@mactech.com](mailto:accounting@mactech.com)

[marketing@mactech.com](mailto:marketing@mactech.com)

[info@mactech.com](mailto:info@mactech.com)

<http://www.mactech.com>

### The MacTech Editorial Staff

**Publisher** • Neil Ticktin

**Editor-in-Chief** • Dave Mark

**Managing Editor** • Jessica Stubblefield

### Regular Columnists

**Getting Started**

by Dave Mark

**QuickTime ToolKit**

by Tim Monroe

**Mac OS X Programming Secrets**

by Scott Knaster

**Reviews/KoolTools**

by Michael R. Harvey

**Patch Panel**

by John C. Welch

**Section 7**

by Rich Morin

**Untangling the Web**

by Kevin Hemenway

**John and Pals' Puzzle Page**

by John Vink

### Regular Contributors

Vicki Brown, Erick Tejkowski,  
Paul E. Sevinç

### MacTech's Board of Advisors

Jordan Dea-Mattson, Jim Straus  
and Jon Wiederspan

### MacTech's Contributing Editors

- Michael Brian Bentley
- Vicki Brown
- Marshall Clow
- John C. Daub
- Tom Djajadiningrat
- Bill Doerrfeld, Blueworld
- Andrew S. Downs
- Richard V. Ford, Packeteer
- Gordon Garb, Sun
- Ilene Hoffman
- Chris Kilbourn, Digital Forest
- Rich Morin
- John O'Fallon, Maxum Development
- Will Porter
- Avi Rappoport, Search Tools Consulting
- Ty Shipman, Kagi
- Chuck Shotton, BIAP Systems
- Cal Simone, Main Event Software
- Steve Sisak, Codewell Corporation
- Chuck Von Rospach, Plaidworks

### Xplain Corporation Staff

**Chief Executive Officer** • Neil Ticktin

**President** • Andrea J. Sniderman

**Controller** • Michael Friedman

**Production Manager** • Jessica Stubblefield

**Production/Layout** • Darryl Smart

**Marketing Manager** • Nick DeMello

**Account Executive** • Lorin Rivers

[adsales@mactech.com](mailto:adsales@mactech.com) • 800-5-MACDEV

**Events Manager** • Susan M. Worley

**International** • Rose Kemps

**Network Administrator** • David Breffitt

**Accounting** • Jan Webber, Marcie Moriarty

**Customer Relations** • Laura Lane, Susan Pomrantz

**Shipping/Receiving** • Dennis Bower

**Board of Advisors** • Steven Geller, Blake Park,  
and Alan Carsrud

All contents are Copyright 1984-2003 by Xplain Corporation. All rights reserved. MacTech and Developer Depot are registered trademarks of Xplain Corporation. RadGad, Useful Gifts and Gadgets, Xplain, DevDepot, Depot, The Depot, Depot Store, Video Depot, Movie Depot, Palm Depot, Game Depot, Flashlight Depot, Explain It, MacDev-1, THINK Reference, NetProfessional, NetProLive, JavaTech, WebTech, BeTech, LinuxTech, MacTech Central and the MacTutorMan are trademarks or service marks of Xplain Corporation. Sprocket is a registered trademark of eSprocket Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders.

**MacTech Magazine** (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are \$47.00 per year. Canadian subscriptions are \$59.00 per year. All other international subscriptions are \$97.00 per year. Domestic source code disk subscriptions are \$77 per year. All international disk subscriptions are \$97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

**POSTMASTER:** Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.



# C o n t e n t s

April 2004 • Volume 20, Issue 4

## INTERACTIVE DEVELOPMENT

### 52 Interactive Development With ActiveDeveloper

*An Introduction to Interactive Development and a review  
of Inter\*ACTIVE - Technology's ActiveDeveloper*

*By Aidan A Reel*

## REVIEWS

### 66 Testtrack Pro 6.1

*Defect tracking for distributed development teams*

*By Sean Whelan*

## REALBASIC CHAT

### 48 Notes from the REALWorld Conference

*REAL Software's first conference revealed  
exciting new directions*

*By Guyren G Howe*

## MAC OS X PROGRAMMING SECRETS

### 14 Shell Game: Calling Shell Commands from Applications

*By Scott Knaster*

## REALBASIC BEST PRACTICE

### 62 Object-Oriented Programming with REALbasic

*Getting the most from a unique Object-Oriented paradigm*

*By Guyren G Howe*

## UNTANGLING THE WEB

### 44 Database Modification with a GUI

*Modifying our MySQL database further with GUI-based editors.*

*By Kevin Hemenway, Windusrtian Tarutaru*

## GETTING STARTED

### 4 Finishing Our First Cocoa App

*By Dave Mark, Editor In Chief*

## APPLESCRIPT

### 25 ESSENTIALS

#### The "NEW" Script Editor

*Copyright 2004 by Benjamin S. Waldie*

## REVIEWS

### 18 Perforce

*Powerful version control for the Mac  
(and all those other platforms)*

*By Paul Pharr, Annapolis, MD*

## PATCH PANEL

### 40 A Look At Panther Server

*Digging past the hype into Apple's latest server OS*

*By John C. Welch*

## QUICKTIME TOOLKIT

### 32 Snow Day

*Developing QuickTime Applications for the iPod*

*by Tim Monroe*

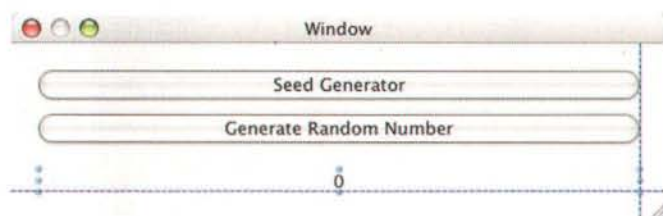


Copyright 2004 by Dave Mark

## Finishing Our First Cocoa App

In last month's column, I told you about my fabulous trip to *Big Nerd Ranch* for some down home Cocoa training with Aaron Hillegass and crew. We then started on our first in a series of Cocoa programs. We used Xcode to create a new project (named *RandomApp*), then used Interface Builder to edit the nib file and create the *RandomApp* user interface.

**Figure 1** shows the final configuration of the window we laid out in Interface Builder. The top button is used to seed the random number generator. Press the second button to generate a new random number. The new number is displayed in the text field below the two buttons. In **Figure 1**, the text field is selected and the blue, dashed lines show the window was being resized to leave the proper amount of space below the text field and to the right of the two buttons and the text field.



**Figure 1.** The final window, showing two buttons and a text field.



GraphicConverter converts pictures to different formats. Also it contains many useful features for picture manipulation.

See **[www.lemkesoft.com](http://www.lemkesoft.com)**  
for more information.

Once the window was laid out to our satisfaction, we selected *Test Interface* from Interface Builder's *File* menu to take the interface for a test drive.

So what's next? Aha! Glad you asked...

### WHAT'S NEXT

Unless you've been patiently waiting with both Xcode and Interface Builder running in the background, chances are good that you saved last month's efforts and have since quit both apps.

OK, let's get back to work.

Launch Xcode and open last month's Xcode project. I saved my copy in my *Documents/Projects/* directory in a subdirectory named *RandomApp*. Find and launch *RandomApp.xcode*.

When the project window appears, open the triangle to the left of the *NIB Files* group to reveal the project's nib file, *MainMenu.nib*. Double click on the nib file to open it in Interface Builder.

### Outlets and Actions

Your next task is to create a subclass of *NSObject* that we'll use to respond to clicks in the *Seed Generator* and *Generate Random Number* buttons. The class will also modify the text field to display the newly generated random number when the *Generate Random Number* button is pressed.

The methods we create to respond to the button clicks are called *actions*. We'll create a unique action that we'll associate with each button.

We'll also need an instance variable that points to the text field so we can change it from the method tied to the *Generate Random Number* button. This kind of object pointer is known as an *outlet*.

When you start a new program, you'll almost always be thinking in terms of actions and outlets. This might be a little fuzzy, but hopefully by the time you get *RandomApp* up and running, this will be a bit clearer.

Let's start by creating our new class. You'll do this in the main Interface Builder window. The folks at *Big Nerd Ranch* like to refer to this window as the "doc" window. Cool with me. The doc window is shown in **Figure 2**. Don't worry too much about

**Dave Mark** is a long-time Mac developer and author and has written a number of books on Macintosh development, including *Learn C on the Macintosh*, *Learn C++ on the Macintosh*, and *The Macintosh Programming Primer* series. Dave's been busy lately cooking up his next concoction. Want a peek? <http://www.spiderworks.com>.



# DISTRIBUTE SECURE SOFTWARE



- > CHOOSE YOUR PATHS.
- > NAIL YOUR TARGETS.
- > REAP THE REWARDS.

**Privilege™**  
SECURE SOFTWARE COMMERCE

SECURE SOFTWARE COMMERCE for ESD, CD-ROM, Casual Sharing and Media-based distribution  
Increased revenues. Reduced costs. Measurable results. That's the power of Privilege. Feel it for yourself.  
For your FREE Evaluation Kit, call 1-800-562-2543 or visit [GoPrivilege.com/YourWay](http://GoPrivilege.com/YourWay).

N. America: 1-800-562-2543, 847-818-3800 or [Priv.us@eAladdin.com](mailto:Priv.us@eAladdin.com) Int'l: +972-3-636-2222 or [Priv.il@eAladdin.com](mailto:Priv.il@eAladdin.com)  
Germany: [Priv.de@eAladdin.com](mailto:Priv.de@eAladdin.com) UK: [Priv.uk@eAladdin.com](mailto:Priv.uk@eAladdin.com) France: [Priv.fr@eAladdin.com](mailto:Priv.fr@eAladdin.com)  
Benelux: [Priv.nl@eAladdin.com](mailto:Priv.nl@eAladdin.com) Japan: [info@Aladdin.co.jp](mailto:info@Aladdin.co.jp)

©2004 Aladdin Knowledge Systems, Ltd. Aladdin, Aladdin Knowledge Systems, Ltd., and HASP are registered trademarks of Aladdin Knowledge Systems, Ltd.

**Aladdin®**  
SECURING THE GLOBAL VILLAGE  
[eAladdin.com](http://eAladdin.com)



what all the different icons mean. We'll explore them all in future columns.

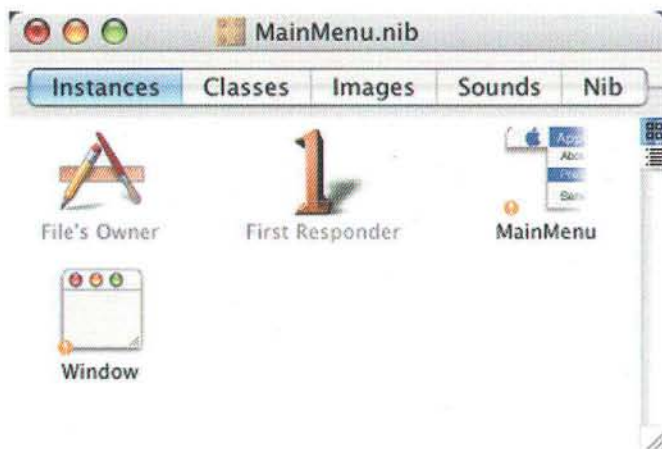


Figure 2. Interface Builder's "doc window".

For now, click on the doc window's *Classes* tab. If you don't see *NSObject* in the leftmost pane (See **Figure 3**), scroll to the left using the scrollbar at the bottom of the doc window. Click on *NSObject* to select it. *NSObject* is the root class for all Cocoa classes. Note that all classes start with a capital letter. When you create your class, you'll follow this convention.

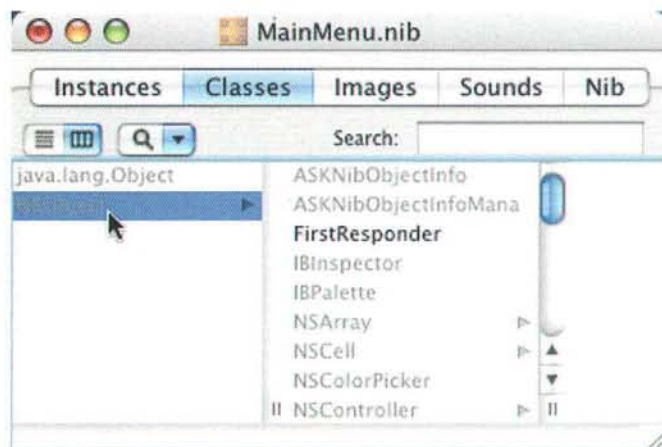


Figure 3. The *NSObject* class selected in the *Classes* tab of the doc window.

To create your *NSObject* subclass, select *Subclass NSObject* from Interface Builder's *Classes* menu. A new class will appear in the second pane. Change the name to *Foo*, then hit *return* to lock in the name change. The name *Foo* should appear immediately after *FirstResponder* in the second column.

Now let's add our outlet and actions to our new *Foo* class.

With the *Foo* class name selected in the second column of the doc window, select *Show Info* from the *Tools* menu to bring up the inspector window. As you can see in **Figure 4**, the

inspector currently shows 0 outlets and 0 actions for the *Foo* class. We'll start off by adding an outlet.

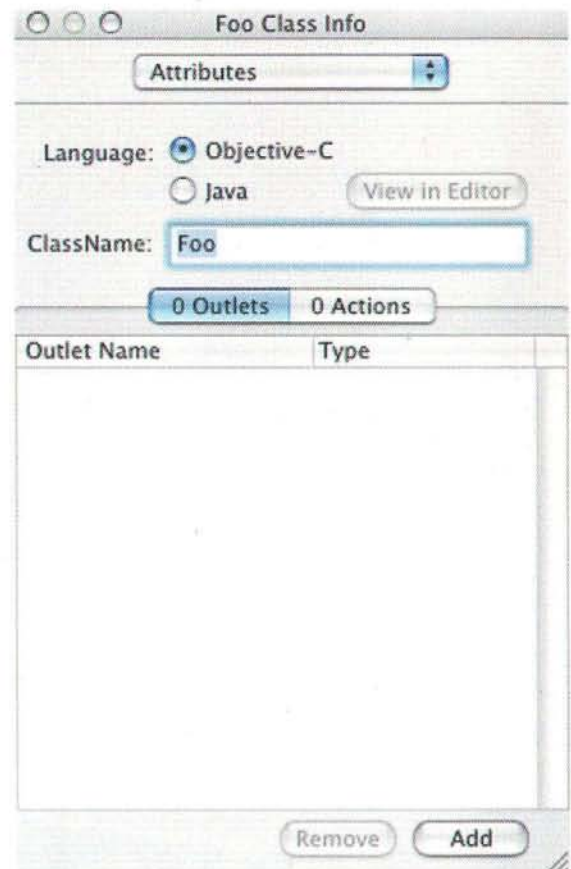


Figure 4. The inspector, showing 0 outlets and 0 actions for the *Foo* class.

Remember, an outlet is an instance variable that is a pointer to an object. Instead of doing all the work in code, we're going to use the point-and-click power of Interface Builder to add the class, then add the outlet instance variable and action methods to the *Foo* class and, ultimately, to generate the source code files that implement the *Foo* class to the Xcode project. Once you get used to this process, you'll never want to create classes from scratch again. Incredibly powerful stuff.

To add an outlet to *Foo*, click on the *Add* button in the lower-right corner of the inspector window (**Figure 4**). Be sure that the *Outlets* tab is selected or you'll be adding an action instead. Once you click the *Add* button, a new outlet will appear. Name the outlet *textField*, then set the outlet's type to *NSTextField* using the popup menu in the second column. Basically, you've just told Interface Builder to add an instance variable to the *Foo* class with the name *textField* and the type *NSTextField*. You've used the interface to do this instead of typing the source by hand.



IN THE SOPHOS ZONE, VIRUSES NEVER INTERRUPT YOUR WORKFLOW



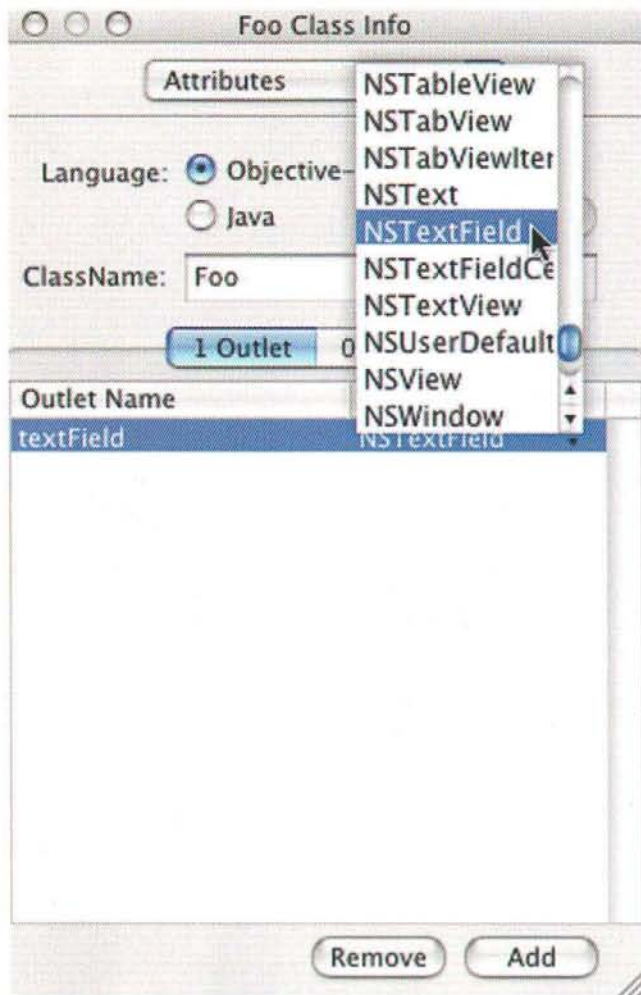
Sophos Anti-Virus provides total protection against all known viruses on Mac OS X. Its GUI enables users to perform an immediate scan of any accessible file, folder and volume, while InterCheck technology keeps users safe by intercepting all file accesses and scanning for viruses in the background in real-time. In addition, every license includes 24x7x365 technical support. In the Sophos Zone, viruses don't stand a chance.

For a free, fully supported evaluation, visit:  
[www.sophos.com/av\\_mac](http://www.sophos.com/av_mac)

[www.sophos.com/av\\_mac](http://www.sophos.com/av_mac)  
Tel 888-216-6703

**SOPHOS**  
engineered for business





**Figure 5.** Set the outlet's type to *NSTextField* from the popup menu.

Now let's add a couple of actions. Click on the *Actions* tab and click the *Add* button. Name the new action *seed:*, being sure to include the trailing colon. The colon is actually part of an Objective C method name. Note that by convention, Objective C method names always start with a lower case letter (as opposed to class names which, as mentioned, start with an upper case letter).

Next, add a second method. Call it *generate:* (yup, remember the colon). When you are done, your inspector window should look like the one shown in **Figure 6**.

If you find your inspector window getting out of sync, it might be because you have clicked on a different Interface Builder object. The inspector always reports on the currently selected object. Click on a window, the inspector shows the window's properties. If you *do* get lost, go back to the doc window, click on the *Classes* tab, scroll all the way to the left, then click on the *NSObject* class, then on the *Foo* subclass. The title of the inspector

window should now be *Foo Class Info*. If the contents of the window do not match **Figure 6**, be sure to select *Attributes* from the popup menu at the top of the inspector window.



**Figure 6.** The inspector window showing my two actions.

### Generate the Source Files

You've now laid out the pieces that make up your *Foo* class. You've got a pointer to an *NSTextField*, as well as a pair of methods you'll want called when the user clicks the *Seed Generator* or *Generate Random Number* buttons.

Your next step is to generate the source code that defines the *Foo* class and add that source code to your Xcode project. Fortunately, Interface Builder can do all this for you with one menu selection.

Make sure that the *Foo* class name is selected in the second column of the doc window. Now, select *Create Files for Foo* from the *Classes* menu. A sheet will appear (See **Figure 7**) that asks you where you'd like to save the files, what type of files to create, and to which target to add the files. Unless you've done some monkeying around, the defaults will probably be just fine. You'll save the files in your main project directory (mine is called *RandomApp*). You'll want to create both a *Foo.h* and a *Foo.m*



file. `Foo.h` is the include file that contains your class declaration. `Foo.m` contains the actual class definition (the seed: and generate: source code, for example).

The target is the specific Xcode build you want these source files to be part of. Since we only have one target, this is an easy choice. Make sure the *RandomApp* checkbox is checked.

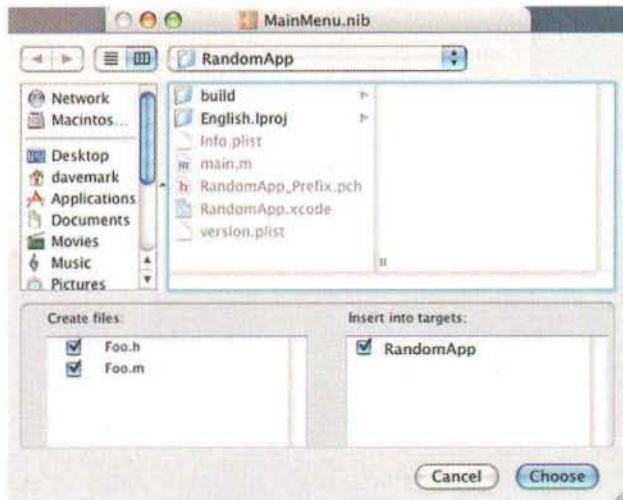


Figure 7. Saving the *Foo* class files.

Click the *Choose* button to add the source files to your project. Notice that two new source code file names appear in your Xcode project window in the *Groups & Files* pane. Typically, the two names will be appended to the list in the *Other Sources* group (See **Figure 8**). Feel free to drag them into another group or subgroup as you like.



Figure 8. The files *Foo.h* and *Foo.m* are added to the *Groups & Files* list.

## Create and Connect a *Foo* Instance

Before we edit the source code, there's just one more task ahead of us. We need to create an instance of the *Foo* class, then connect the outlet and actions to the appropriate objects. This will become clearer in a moment.

Go back into Interface Builder.

In the doc window, click on the *Foo* class name in the second column. Select *Instantiate Foo* from the *Classes* menu. To see your new object, click on the *Instances* tab in the doc window.

As you can see in **Figure 9**, a new icon has appeared. In the doc window. Note the tiny explanation point in the lower left corner of the *Foo* icon. If you hover your cursor over the exclamation point, a tool-tip will appear saying "Unconnected outlet(s) (textField)." We've declared an outlet named *textField* but we haven't connected it to a text field. Let's fix that.

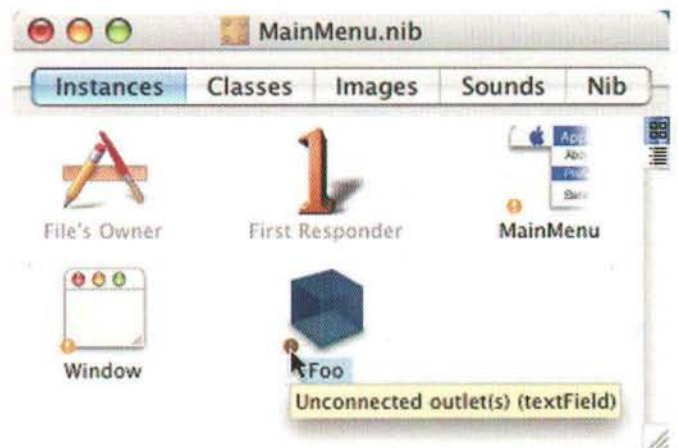


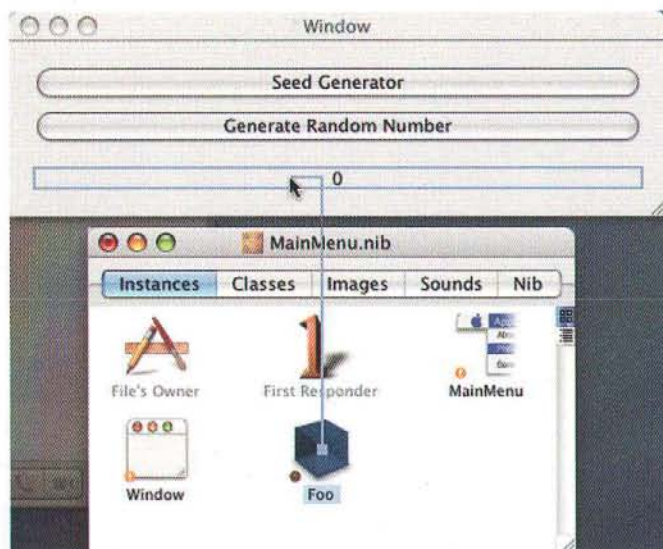
Figure 9. The *Foo* instance in the doc window. Note the message about *Unconnected outlet(s)*.

Control-drag from the *Foo* instance to the text field below the two buttons in our application window. As you can see in **Figure 10**, when you start the control-drag, a little square will appear in the *Foo* icon, then a line will follow your cursor as you drag to the text field. Once the text field is highlighted with a surrounding rectangle, you can release the mouse button.

To complete the connection, click the *Connect* button in the lower-right corner of the inspector window.

The purpose of this connection is to fill the *Foo* object's *textField* instance variable with a pointer to the window's text field so we can modify the contents of the text field in our source code.





**Figure 10.** Control drag from Foo to the text field below our two buttons.

Our next step is to connect each of our two buttons to the appropriate method using the same technique.

Control-drag from the *Generate Random Number* button to the Foo instance. In the inspector window, you should see a list of Foo methods to choose from. Click on *generate:* and then click the *Connect* button (See **Figure 11**). You can also double-click on *generate:* instead.

If your inspector window gets out of sync, be sure that *Connections* is selected from the popup at the top of the window, then make sure that the *Target/Action* tab is selected. Finally, be sure that you dragged from the *Seed Generator* button to the Foo instance.

Next, let's connect the *Seed Generator* button to the *seed:* action. Control-click from the *Seed Generator* button to the Foo instance in the doc window. Next, select *seed:* and click the *Connect* button.



**Figure 11.** Click the *Connect* button to connect the *Generate Random Number* button to the *generate:* method.

That's it! **Save your changes.** This last step is *very important* as we are going to switch over to Xcode and we want to be sure we are dealing with the latest version of the .nib file.

### Editing Your Source Code

In most programming projects, editing the source code is by far the biggest step to building your project. Because of the power of Interface Builder and the reusability of all the existing Cocoa objects, our source code changes will be pretty minimal.

Let's take a look at Foo.h.

Find Foo.h in the *Groups & Files* pane. Click on it. If the source code does not appear in an editing pane, select *Show Embedded Editor* from the *View* menu.

Here's the source code that Interface Builder placed in Foo.h:

```
/* Foo */

#import <Cocoa/Cocoa.h>

@interface Foo : NSObject
{
    IBOutlet NSTextField *textField;
}
- (IBAction)generate:(id)sender;
- (IBAction)seed:(id)sender;
@end
```

The *#import* statement is basically a *#include* that avoids the recursive effects of including a file that includes you.

The *@interface* statement declares the Foo class as a subclass of *NSObject*. There is a single instance variable, *textField*. Instance

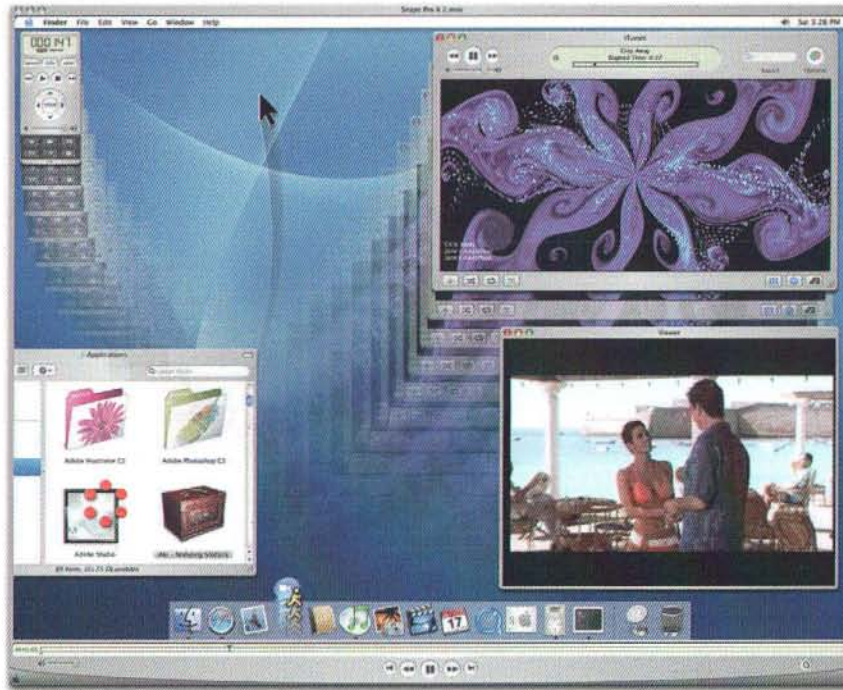
**MacTech®**  
M A G A Z I N E

Get MacTech delivered to your door at a price **FAR BELOW**  
the newsstand price. And, it's **RISK FREE!**

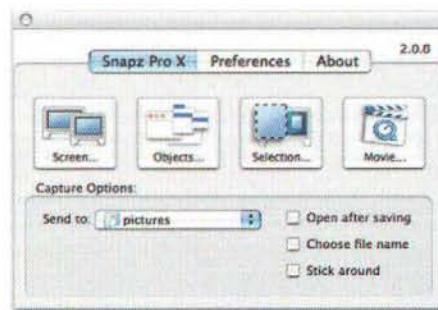
**Subscribe Today!**  
**www.mactech.com**



If a picture is worth a thousand words,  
imagine how priceless a movie would be...



Snapz Pro X 2.0 allows you to effortlessly record anything on your screen, saving it as a QuickTime® movie that can be emailed, put up on the web, or passed around however you please.



Why take a static screenshot when Snapz Pro X 2.0 makes creating a movie just as easy?  
Snapz Pro X 2.0 does that, and so much more -- what a difference a version makes!  
Download a free demo version from our web site today and see for yourself:

<http://www.AmbrosiaSW.com/>



**Snapz Pro X 2.0**  
**AMBROSIA®**  
**SOFTWARE INC**



Snapz Pro X

Snapz Pro X, Ambrosia Software, Inc., and the Ambrosia Software logo are registered trademarks of Ambrosia Software, Inc.



variables are declared inside the curly braces, Methods are declared after the curly braces and before the *@end* statement.

You don't need to change a line of this code. Interface Builder did all the work for you. Let's edit Foo.m, where the real action is. Here's what Foo.m looks like before you change it:

```
#import "Foo.h"

@implementation Foo

- (IBAction)generate:(id)sender
{
}

- (IBAction)seed:(id)sender
{
}

@end
```

Here's the edited version:

```
#import "Foo.h"

@implementation Foo

- (IBAction)generate:(id)sender
{
    int generated;

    generated = (random() % 100) + 1;

    [textField setIntValue:generated];
}

- (IBAction)seed:(id)sender
{
    srand( time( NULL ) );
    [textField setStringValue:@"Generator seeded"];
}

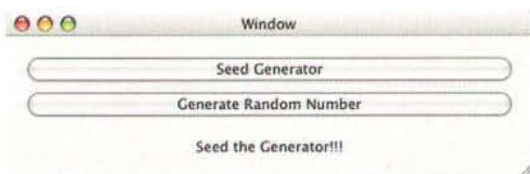
- (void)awakeFromNib
{
    [textField setStringValue:@"Seed the Generator!!!"];
}

@end
```

Basically, you are adding 3 lines to the generate: method, 2 lines to the seed: method, and the 4 lines that make up the awakeFromNib: method. I added awakeFromNib: so the text field would have a reasonable setting at startup.

Don't worry too much about the code itself. The key here is that with 9 pretty simple lines of code and some time invested in Interface Builder, we've created a running application that does something interesting.

Build and run the application by selecting *Build and Run* from the *Build* menu. **Figure 12** shows RandomApp when it starts up. Click the *Seed Generator* button, then click *Generate Random Number* to start generating random numbers. Hey, it works!



**Figure 12.** RandomApp in action.

## SOME RAMBLINGS BEFORE I GO

Some unrelated blathering before I leave. Sort of a static blog, if you will.

Deneen and I just got a Prius. To me, this is the car of the future. Today, I really, really love this car. It is a hybrid vehicle with a gas engine and an electric motor. The Prius switches between the two as needed. Gets extremely high gas mileage and has very low emissions. Worth checking out.

If you get a sec, go to <http://www.spiderworks.com> and check out my new project. It consumes me.

Also, I want to stand up on my chair and applaud O'Reilly for publishing Wil Wheaton's short story collection, *Dancing Barefoot*. I love that they did this. Wil Wheaton is an actor, played Wesley Crusher in a past life and, most importantly, has an exceptionally entertaining web site:

<http://www.wilwheaton.net>

Check out the site, check out the book.

## OH YEAH – WWDC IS JUST AROUND THE CORNER

Can you believe it? WWDC is just a couple of months away. Just like last year, this year's Worldwide Developers Conference will be in San Francisco from June 28th through July 2nd. Apple has reported that they have close to 10 million active Mac OS X users and over 10,000 native Mac OS X applications. These numbers show incredible adoption growth since last year's conference and that's good news for developers.

This year's conference offers 7 tracks: Application Technologies, Development Tools, Enterprise IT, Graphics and Media, Hardware Technologies, OS Foundations, and QuickTime Digital Media. Look for the QuickTime track to mix in integration of Pro Apps (DVD Studio Pro, for example) and to cover creation of audio loops for Soundtrack, GarageBand, Logic, etc. This one is definitely on my short list!

Read all about the conference here:

<http://developer.apple.com/wwdc/features.html>

Note that there is a discount if you buy your ticket by April 30th.

## TILL NEXT MONTH...

I am having a *great* time writing about Cocoa. Expect this nonsense to continue. If you insist on homework, find the *Property List Editor* (/Developer/Applications/Utilities/) and use it to open (a copy of) your .nib file. Fascinating.

See you in the future...



# WIBU-KEY Software Protection

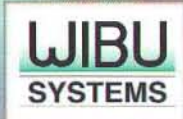
Only hardware can securely protect software



- Common API across all platforms and hardware
- Hardware-based code and data encryption
- Field-upgradable and reusable hardware
- The most cost-effective network licensing solution available
- The only system to employ RID/RED and AXAN security
- Engineered and manufactured to exacting ISO9001 standards

**Test the WIBU-KEY Protection Kit**  
**Call 800-986-6578**  
**[sales@griftech.com](mailto:sales@griftech.com)**

**The Key is in Your Hands!**



WIBU-SYSTEMS USA, Inc.  
Seattle, WA 98101  
Email: [info@wibu.com](mailto:info@wibu.com)

**[www.griftech.com](http://www.griftech.com)**  
**[www.wibu.com](http://www.wibu.com)**

Protection Kits also available at:

Belgium [wibu@impakt.be](mailto:wibu@impakt.be), Czech Republic [info@elbacom-slo.com](mailto:info@elbacom-slo.com), Denmark [lean@danbit.dk](mailto:lean@danbit.dk), France [info@neol.fr](mailto:info@neol.fr), Hungary [info@mrsoft.hu](mailto:info@mrsoft.hu),  
Japan [info@suncarla.co.jp](mailto:info@suncarla.co.jp), Jordan, Lebanon [starsoft@cyberia.net.lb](mailto:starsoft@cyberia.net.lb), Korea [dhkimm@wibu.co.kr](mailto:dhkimm@wibu.co.kr), Luxembourg [wibu@impakt.be](mailto:wibu@impakt.be),  
Netherlands [wibu@impakt.be](mailto:wibu@impakt.be), Poland [info@elbacom-slo.com](mailto:info@elbacom-slo.com), Portugal [dubite@dubit.pt](mailto:dubite@dubit.pt), Slovakia [info@elbacom-slo.com](mailto:info@elbacom-slo.com),  
Slovenia [info@elbacom-slo.com](mailto:info@elbacom-slo.com), Thailand [preecha@dptf.co.th](mailto:preecha@dptf.co.th), United Kingdom [info@codework.com](mailto:info@codework.com), USA [sales@griftech.com](mailto:sales@griftech.com)



By Scott Knaster

# Shell Game: Calling Shell Commands from Applications

The coolest thing about Mac OS X is the skillful grafting of the lovely and attractive Aqua graphical user interface onto the geeky but sturdy Unix stuff underneath. Applications in OS X run on a Darwinian layer of software that you can access via Unix shells in Terminal, also known as the command line. This gives you the ability to lift the hood on an application and see what's going on underneath. You can observe an application's behavior, change its settings, or even fool around with its workings, if you know what you're doing *or* you don't mind the possibility of breaking things.

The commands you can run in the shell are very powerful. Sometimes it's handy to use a shell command from within an application, giving you an ideal combination of friendly user interface and utter command line power. In this month's column, we'll explore how to run shell commands from a Cocoa application. By the time you finish this column, you will be a master (or at least someone capable) of using shell commands from your lofty Cocoa programs.

## HOW DID I GET HERE?

What are shell commands, and where do they come from? Shell commands are instructions you type in Terminal to make things happen. For example, you type `ls` when you want a directory listing, `kill` to end a process, `cp` to copy a file, and so on.

Shell commands aren't built into the shell, with very few exceptions. They're actually executable programs stored in well-known (to the shell) directories. When you type the name of a command, the shell searches this set of directories for a file with the same name as the command. When it finds an executable with the sought-after name, the shell runs it. If there's no file by that name, you get the disheartening message **Command not found**.

The list of directories that hold shell commands is kept in a shell variable name `PATH`, so named because it holds the search path for commands. You can see the value of the `PATH` variable by typing a shell command: `echo "${PATH}"`. Here's what I got when I typed that command:

```
[neb:~] scott% echo ${PATH}
/bin:/sbin:/usr/bin:/usr/sbin:/Developer/Tools
```

The command output shows that the shell will look for commands in 5 directories: `/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`, and `/Developer/Tools`. If we take a look in those directories, we should see some familiar command names living there. And sure enough, we do. For example, here are the files in `/bin`:

```
[neb:~] scott% ls /bin
[      date      expr      mv      rmdir      test
bash      dd      hostname  pax      sh      zsh
cat      df      kill      ps      sleep      zsh-4.1.1
chmod      domainname  ln      pwd      stty
cp      echo      ls      rcp      sync
csh      ed      mkdir      rm      tcsch
```

There are 33 commands in this directory, and hundreds more in the others. The `/usr/bin` directory alone includes over 600 commands.

The search path is initialized when the shell starts. The initial value for this and other shell variables is set by a configuration file named `.tcshrc`. This file is used with the `tcsh` shell, which is the default shell in OS X 10.3. If you switch to a different shell, such as `bash`, you'll have a different configuration file. Commands in `.tcshrc` are executed when the shell starts up. Let's take a look and see what's in `.tcshrc`:

```
[neb:~] scott% cat .tcshrc
setenv PATH "${PATH}:/Developer/Tools"
```

In this case, the `.tcshrc` file adds `/Developer/Tools` to the search path. This line gets tacked on to your configuration file when you install Xcode. If you want to add other directories of commands to the search path, you can put more `setenv` commands in your shell configuration file.

## TASK IT

A shell command is just an executable file. When the system runs an executable, it creates a process. Because Cocoa likes things best when they're objects, Cocoa provides the `NSTask`

**Scott Knaster** attended Cocoa Bootcamp at Big Nerd Ranch (<http://www.bignerdranch.com/classes/cocoa1.shtml>). He did not go Cocoa Loco but he liked it very much. Scott counts the days until pitchers and catchers report.





**ENGINEERING  
BUSINESS  
SOLUTIONS**

**- SINCE 1989 -**

**PREMIER REPORT SOLUTIONS FOR MAC OS X**

**VVI®**

[www.vvi.com](http://www.vvi.com)

[info@vvi.com](mailto:info@vvi.com)

**888-VVI-PLOT**



class in the Foundation framework for handling processes. A process has an execution environment that includes its current directory, environment variables, and other values. You can create an `NSTask` object in your application and use it to control a shell command process.

It's going to be darned easy to call our first shell command. We'll only need a few methods of `NSTask`:

- `setLaunchPath` specifies the file path of the command we're going to call from our application.
- `setArguments` supplies any arguments that we would normally pass via the command line. Technically, this method isn't always necessary, because some commands don't take any arguments. But most of them do, most of the time.
- `launch` executes the command and creates a new process from it.

That's all there is to it. What could be simpler? Of course, you should call `alloc` and `init` when you create the `NSTask`, and `release` when you're done.

In order to keep our example as basic as possible, we'll choose the `open` command, which simply opens a file or directory. For our example, we'll have it open the Applications folder to remind us of all the cool apps we're privileged to have on our Macs.

OK, let's make the application. In Xcode, choose New Project and create a Cocoa application. Name it whatever you like – a nice name, like *Melanie*, or *Commando*. Double-click *MainMenu.nib* to open it in Interface Builder. In the application window, add an `NSButton` object. Shrink the window down and put the button in the center. Name the button "Show Apps". Your screen should look something like **Figure 1**.

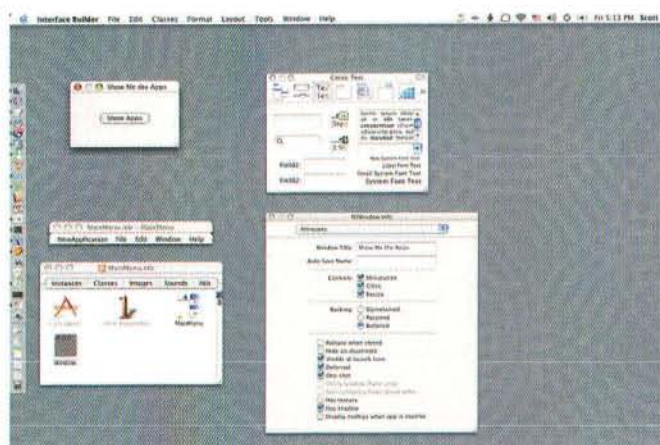


Figure 1. Laying out our modest application in Interface Builder.

NeXT, click the **Classes** tab, select `NSObject`, and press return to create a subclass. Rename the subclass `AppController`. In the Inspector (which you can open with Tools ⌘ Show Info), click the **Actions** tab, then click **Add** to create an action named

`doCommand`. Create the files for the class (Classes ⌘ Create Files for `AppController`) and make an instance (Classes ⌘ Instantiate `AppController`).

Now it's time for the hookup. Connect the button to the `AppController` object by Control-dragging from the button to the `AppController` (in the *MainMenu.nib* window). Double-click the `doCommand` action to complete the wiring. Your screen in IB (which is what the cool kids call Interface Builder) should now look like **Figure 2**.

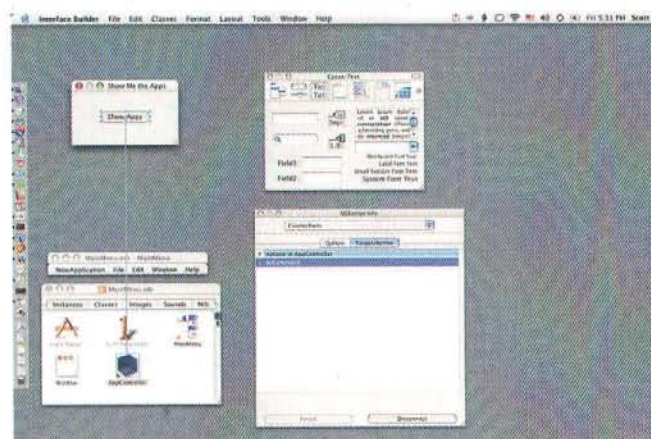


Figure 2. Connect the button to the application controller.

Turning our short attention span back to Xcode, it's time to write the code. Double-click `AppController.m` and type in the `doCommand` method.

```
#import "AppController.h"

@implementation AppController

- (IBAction)doCommand:(id)sender
{
    NSTask *theProcess;
    theProcess = [[NSTask alloc] init];

    [theProcess setLaunchPath:@"/usr/bin/open"];
    // Path of the shell command we'll execute
    [theProcess setArguments:
     [NSArray arrayWithObject:@"/Applications"]];
    // Arguments to the command: the name of the
    // Applications directory

    [theProcess launch];
    // Run the command

    [theProcess release];
}

@end
```

First, we create and set up the task object by calling `NSTask alloc` and `init`. Then, once we've instantiated the object, we tell OS X where to find the executable by calling `setLaunchPath`. In this case, it's in `/usr/bin`. (How did I know it was in that directory and not another? I looked for it. Remember from our earlier exercise that shell commands are generally in one of five directories: `/bin`, `/sbin`, `/usr/bin`, `/usr/sbin`, or `/Developer/Tools`. You can find the location of any shell command by calling the, er,



shell command **which**. And to learn the details of using **which**, you of course go to the shell and type **man which**. Ah, Unix, where the fun never ends.)

After setting up the launch path, we call **setArguments** to say what should be passed to the command when it's called. Here, the only argument we need is what should be opened – the Applications folder – so we pass **"/Applications"**. When we call **launch**, the command runs, the Finder comes to the front, and the Applications folder opens up. We did it!

### INCREASING OUR OPENNESS

The command we used, **open**, has a bunch of options that make it a very versatile tool, and you can use any of them when you call it from an application. For example, you can open any file just by specifying its path. If the file is a document, it will be opened with the default application. You can also use **open** to view a web page in the default browser. Just don't forget the protocol, such as **http**. Do it like this:

```
[theProcess setArguments:
 [NSArray arrayWithObject: @"http://www.mactech.com"]];
```

The **open** command has an option, **-a**, that lets you open a file with a program that's not the default. You can use this when you call **open** from the command line or from an application. In the shell, it would look like this:

```
open -a /Applications/TextEdit.app /Users/Scott/grape.doc
```

In this example, **grape.doc** will open with **TextEdit** instead of **Microsoft Word** as Bill intended. To achieve this, we're passing multiple arguments to **open**. The technique for calling **setArguments** is a little different when you have more than one argument. Because **setArguments** actually takes an **NSArray**, you have to do something like this:

```
[task setArguments:[NSArray arrayWithObjects:
 @"-a",
 @"/Applications/TextEdit.app",
 @"/Users/Scott/grape.doc",
 nil]];
```

Here we call **arrayWithObjects** – that's objects, plural – compared to earlier when we called **arrayWithObject**, singular, with only one parameter. The arguments are separated by commas. And surprisingly, the **-a** is a separate argument from the application name that follows it. Another wacky thing to remember about **arrayWithObjects** is that you have to add a **nil** after your last real object.

### USERS CAN BE CHOOSERS

Our example uses the **open** command to show the Applications folder in the Finder. That's a fine demonstration of the ability to call a shell command from a Cocoa application, but really, how many times can you look at the same folder? Let's make it slightly more interesting by allowing the user to decide what to open.

To do this, we'll go back to Interface Builder and add an **NSTextView** to the window. We'll name the new text view **fileName** – a fine, functional name. Then we'll add an outlet to the **AppController** object for **fileName** and introduce the objects to each other by Control-dragging. We save our work in IB and then head over to Xcode.

We need to make a simple change to **AppController.m**. Find the line where we call **setArguments** and change it to:

```
[theProcess setArguments:
 [NSArray arrayWithObject: [fileName string]]];
```

This line now extracts the text from **fileName** and uses it to set the arguments for our impending shell command call. Our remodeled application window is pictured in **Figure 3**.



**Figure 3.** Our application now allows the user to enter an item to be opened. This product is now almost ready to ship, and t-shirts will be available soon.

As we discussed previously, this technique works when you have one argument, but not multiple arguments. If you're looking for something to do, a fun exercise would be to add another text view to the window and use it to let the user specify the other argument, an application to use for opening the file.

### MOVING RIGHT ALONG

We accomplished our goal of calling a shell command from Cocoa, but we only scratched the surface, with neatly trimmed and manicured nails, of what you can achieve with **NSTask**. Next month we'll continue messing around with **NSTask** to see what other fun we can have.

**MacTech**<sup>®</sup>  
M A G A Z I N E

Get MacTech delivered to your door at a price **FAR BELOW**  
the newsstand price. And, it's **RISK FREE!**

**Subscribe Today!**  
**www.mactech.com**





By Paul Pharr, Annapolis, MD

# Perforce

## Powerful version control for the Mac (and all those other platforms)

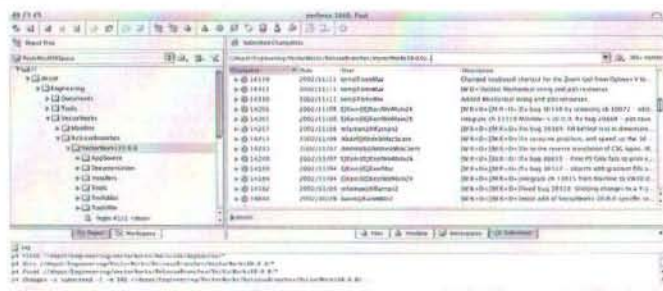


Figure 1 – P4V, the MacOS X client GUI

### INTRODUCTION

When I was in engineering school, my software engineering professor made a point of saying to the undergraduates, "Use the tools you have." This bit of real-world advice was useful both because the tools we use are rarely the enabling factor for the success of software projects, and also because individual engineers are rarely in a position to dictate the tools to be used on a project. Mac developers are especially aware of limitations imposed on their choices by factors beyond their control. With this in mind, Mac developers are fortunate to have Perforce as an available and well-supported option when shopping for a revision control system.

### MY PERSPECTIVE

I work for Nemetschek North America – formerly Diehl Graphsoft – makers of MiniCAD & VectorWorks CAD software. We have a development environment with about 40 engineers interacting with our source code base through Perforce – a tool that we selected three years ago to replace Microsoft's Visual SourceSafe. Before that (a long time before) we used MPW's Projector from Apple for version control. I will use our SourceSafe experience as a point of comparison, and I'll mention CVS comparisons if I happen to know, but I have never used

CVS in a production environment. I will try to use Perforce terminology where possible and clarify it where necessary. One exception to this is that I will use the term "check-out" in some cases to indicate the functionality known in Perforce terms as "Open for Edit" because it makes the description more accessible for non-Perforce users.

### WHAT IS VERSION CONTROL

I think it's a safe bet that anyone who works with other developers on a team of any size has a pretty good idea of the basic elements of version control, but to get everyone on the same page, I'll outline some of the basic features. All version control systems allow multiple engineers to work within the same code base by serving as a file librarian and tracking individual users' work on the files that are being modified. They maintain a history of the changes made to files within the system so that earlier versions of the files can be retrieved and differences between versions can be displayed. Version control systems also provide a mechanism to track some additional data about the changes such as who made a change, a description of what was done, and when it happened. Beyond this very basic set of functionality, the capabilities of various systems diverge.

### SYSTEM CAPABILITIES

Perforce, produced by Perforce Software in Alameda, California, is a modern and full-featured version control system intended to be the main repository of all of a software project's files, structure, and history. It has a feature set which can look similar to that of CVS, PVCS, SourceSafe, or ClearCase. With such products, however, a high level outline of the feature set often leaves out a lot about how the product will actually work in a given development environment. Perforce differentiates itself in the following ways:

- Robust
- Fast & Efficient
- Automated merging
- Inter-File Branching Model
- Atomic change submission
- Low administration overhead

**Paul Pharr** manages the ongoing software development of the VectorWorks family of CAD applications at Nemetschek North America. You can reach him at [pharr@nemetschek.net](mailto:pharr@nemetschek.net).



I'll cover each in some detail, and then describe some of the Mac-specific tools and functionality. But first, I'll introduce some basic concepts and terminology.

### PERFORCE BASICS

Understanding Perforce involves a few concepts that I'll cover from a user's point of view so they can be used as a point of reference for examples that follow. Perforce is a client-server system in which the main database runs on a single central server and clients connect using a TCP/IP based protocol to interact with the server. The server can be run on or MacOS X, Windows NT/XP, or various flavors of Linux and Unix. Command line client software is available for almost any conceivable platform, whereas more mature GUI client implementations are available for fewer platforms. Windows has the most mature GUI client software called P4Win - implemented as native Windows code. Mac, Windows, and Linux share a more recently introduced, but very full featured client called P4V (short for Visual) which is about a year old. It is implemented using the QT cross-platform toolkit and is fast and reliable with a native looking GUI on the platforms it supports. The second major release of P4V is in late beta and is what I used on the Mac while working on this review.

Perforce is set up with user accounts for those that will be accessing the system. The "depot" - Perforce's term for the main hierarchy of files under version control - is populated with the files that make up the development projects of the company or department using Perforce. Each user can have one or more client workspaces, which are each associated via preferences maintained on the server with a particular root path on their development machine. Users can have as many workspaces on one or more machines as they find useful. In normal use, the user will keep a copy of some part of the overall depot on their local file system. They will update their workspace files with changes made to the depot by others (described by Perforce as "Sync-ing"), edit files within their local workspace, and submit changes back to the depot.

From this description, Perforce is similar to other version control tools available. Now we'll look at the details that differentiate Perforce.

### PERFORCE IS ROBUST

Keeping source code safe is one of the highest priorities of a software developer, and it's good to know that the tool that is most responsible for the safety of your code places a high priority on maintaining a robust repository. Perforce is architected to facilitate recovery if disaster strikes, but is implemented so well that recovery is rarely if ever necessary.

Perforce is a client-server system in which all client interaction takes place via a TCP/IP connection to a single centralized server. This eliminates a plethora of potential problems compared to systems such as SourceSafe where multiple clients access database files through a shared file system. This architecture gives the server responsibility for

recording changes to system data in a way that allows full recovery should disaster ever strike. Perforce uses an industrial strength database for its metadata and provides for checkpointing and journaling, thereby allowing full recovery from most disaster scenarios. Source code files in Perforce are stored using industry-standard formats for reverse-delta storage, compression, and Mac resource file encoding allowing recovery of their content even if Perforce's databases were completely deleted.

The system also has ample tools to assure you that everything is working as expected. Every revision of every file is given an MD5 hash which is stored in the database and it is straightforward to ask the server to verify that every checksum matches for all files and revisions stored in Perforce. It's an easy and common practice for Perforce sites to regularly verify that every revision in the system is corruption-free.

Knowing that Perforce is built with recovery in mind gives you the comfort of sleeping well at night, but productivity is still compromised if you experience frequent problems. Perforce has an outstanding reputation in this regard among its customers, and our site is certainly evidence of their high reliability. In our three years of using Perforce heavily, we have seen only one server crash - related to differencing revisions of a very large file with very long lines. Perforce support worked with us to carefully but quickly identify the problem and a workaround. They had isolated and fixed the cause and issued a public update to their entire product line within a week of the problem report.

Our use of the product has also confirmed for us that it is virtually impossible for any kind of client failure to cause a database or file corruption on the server. We have occasionally seen bugs in the client software which affect specific features, but they have never had material impact on the overall robustness of Perforce, and have, for the most part, been resolved by a subsequent release of the software. Perforce is, as a whole, at least as robust as any other software we use.

A final factor in robust system performance is that Perforce provides outstanding support - especially in case of emergency. I have heard of a handful of cases where a Perforce server was compromised by hardware failure, but have never heard of a significant loss of data. The consensus in the Perforce user community is that they will do anything in their power to maintain the robust reputation of their software.

Every one of these points stands in stark contrast to the experiences we had with SourceSafe, where we would suffer from file corruptions on a weekly basis, and more significant system corruptions every few months. There were no mechanisms to prevent this or aid recovery. The analysis tool always complained of dire corruption, but provided no means of fixing it. Support was non-existent. We felt like we could lose our entire database at any moment. My impression is that CVS is much better in this respect, but support can still be very hard to come by.



## PERFORCE IS FAST & EFFICIENT

Speed is not often considered a feature of software, but in the world of revision control where individual operations can involve the inspection or transfer of tens of thousands of files, you will soon come to realize which operations are doing more work than they should – and Perforce prides itself on having built efficiency into the system from the ground up.

The database used by the server that I mentioned as a key element of the system's reliability also has dramatic impact on the speed of most normal operations. I'll use updating or syncing a client's source code as an example. In Perforce, the central database keeps a record of every file revision held by every workspace. If you ask to sync to the latest revision of a set of files you don't have in your local workspace, then the server is forced to send you everything, which can be time consuming. During normal work, however, you will usually already have the current revision of most of the files you're working on. In that case, Perforce will only need to send you the files that have changed since your prior sync operation, and it can determine this with a very fast query on an indexed database without inspecting anything on the client machine's file system. A sync operation on a project with 10,000 files typically takes a few seconds, unless it is very out-of-date. The longest sync operations I routinely see are a couple of minutes. SourceSafe and CVS have no provision for optimizing this common operation and will typically exhibit performance corresponding to the total number of files in the hierarchy being updated. We would often wait 10-15 minutes using SourceSafe, whereas Perforce is almost always done in seconds.

As an example, after a week of vacation, I synced our main development branch in just over 2 minutes and got 3500 new files of 18,600 total files in the branch. I synced a maintenance branch and got 5 new files of 10,000 total in about 3 seconds. Most Perforce operations are similarly efficient, including merging changes between branches. Another example - It took about 15 seconds to list all 243,000 files in our depot to a text file using "p4 files //depot/... > filelist.txt"

An obvious benefit of this efficiency is that off-site work becomes feasible. Three of our users are on another continent connected by a 128kbps internet connection. They certainly need to adjust their work habits to account for the lower bandwidth, but not by much. We have never sent them a code snapshot, and they have never been at our site. Nevertheless, they have the same level of project interaction as our local users. Better still, working over a cable-modem sized pipe for local users connecting from home rarely feels much slower than the 100Mb/s switched Ethernet at the office. Perforce also has a recently introduced remote caching server called Perforce Proxy intended to speed up access for an entire remote site. (Our remote site performance without using Perforce Proxy has been good enough that we have decided not to use this tool yet.)

## PERFORCE AUTOMATES MERGING

One of the primary benefits of version control is that it enables concurrent development among engineers. The success of this in a production situation varies depending on the extent to which the tools have been refined, and Perforce does this very well. Merging or resolving differences is also an area that has seen marked improvement in the MacOS clients in recent releases – especially P4V, the new MacOS X GUI client.

There are two situations where you can be exposed to the need to resolve differences. The first is during normal development when you are working on the same set of files as someone else. The second is during multi-branch development where one branch has changes which need to be moved or propagated to another branch. For the sake of this discussion, I'll keep it simple and talk about the first case, but bear in mind that all resolve functionality is pretty much the same regardless of whether you are checking in a small file change to the project you are working on or doing a large inter-branch merge.

For example, you and another developer both check-out and begin editing a file at the same time, but your changes are more extensive, and take longer. You attempt to submit your changes and find that the latest revision of the file you've been editing is newer than the one you started with because the other engineer submitted her changes first. Perforce provides a great deal of control over this process using their resolve functionality. Whenever you ask Perforce to update a file using another version of that file, it uses the always-present database to determine what kinds of changes might be coming across and what kind of action may be necessary. It knows if you have opened a file for editing operations, so if you ask to sync to a newer revision of that file, you may need to resolve potential conflicts between the changes made. Similarly, if you try to submit changes to a file that has been changed in the depot by someone else, you may need to resolve differences. This is the case described above, and there are a number of resolve options available to the user.

After any operation that can encounter conflicting differences, the Perforce GUI indicates files that need to be resolved with a special icon. First, you might try an "Automatic resolve" in which changes made by either engineer will be merged to the result file as long as the changes do not overlap. This typically works and the file is merged, but if it fails because of overlapping changes, you will want to interactively merge the changes. When you merge interactively, you are given the opportunity to review all of the changes made by yourself and the other engineer, and choose which to use. The conflicting changes are the most interesting, as they are the ones that require some modification to the code to preserve the original intent of each change in the final merged file. Perforce will typically handle all but the conflicts automatically, leaving only the work that benefits most from direct user interaction.

If necessary, Perforce will apply all of these operations to very large sets of files at once. My company routinely uses feature branches to accomplish large development projects,



32  
< 1 year  
2 years >  
\$64



## INTERVIEWS

TAPPING INTO THE WORLD OF CELEBRITIES AND THEIR MACS, ONLY MACDIRECTORY OFFERS EXCLUSIVE INTERVIEWS. GET A CLOSE AND PERSONAL VIEW FROM SARAH JESSICA PARKER, STING, STEVE JOBS, MADONNA, HARRY CONNICK JR., GEORGE LUCAS, JENNIFER JASON LEIGH, STEVE WOZ AND OTHER LEADERS IN THE MAC COMMUNITY.



## FEATURES

DESIGNERS, WRITERS, MUSICIANS, BUSINESS LEADERS & OUR TECHNICAL EXPERT TEAM OFFER THEIR OWN PERSONAL INTERPRETATION OF THINGS THAT ONLY THE MAC SYSTEM CAN DELIVER. FEATURING OVER 240 PAGES OF REVIEWS, INTERVIEWS, NEWS, INSIGHTS, TRENDS AND THE LARGEST MACINTOSH BUYERS' GUIDE INCLUDING OVER 5,000 MAC PRODUCTS AND SERVICES.



## CULTURE

MACDIRECTORY TAKES YOU TO THE WILDEST CORNERS OF THE WORLD AND UNCOVERS HOW MACINTOSH COMPUTERS ARE BEING USED BY OTHER CULTURES. TRAVEL TO JAPAN, AUSTRALIA, GERMANY, BRAZIL, INDIA, RUSSIA & LEARN MORE ABOUT APPLE'S CULTURAL IMPACT AROUND THE GLOBE.

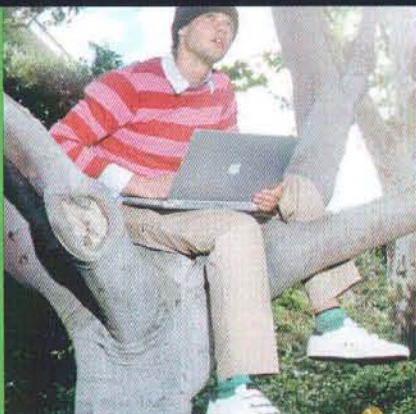
# MacDirectory

BEYOND ANY MACINTOSH MAGAZINE. SUBSCRIBE.

< Subscribe

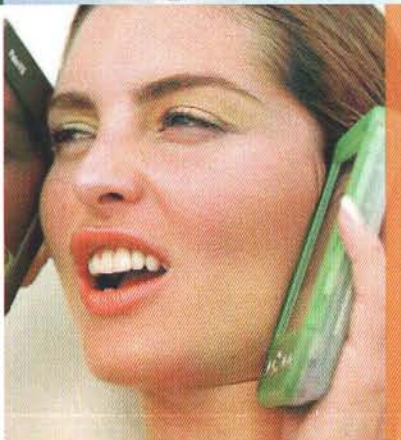
[www.macdirectory.com/mw.html](http://www.macdirectory.com/mw.html)

SEND CHECK OR MONEY ORDER TO:  
MACDIRECTORY SUB DEPT.  
326 A STREET, 2C  
SOUTH BOSTON, MA 02110



## REVIEWS

FIND OUT ALL YOU NEED TO KNOW ABOUT THE LATEST MAC PRODUCTS INCLUDING THE HOTTEST MAC OS SOFTWARE AND HARDWARE.



## WIN!

SUBSCRIBE TO MACDIRECTORY AND YOU WILL AUTOMATICALLY ENTER OUR SWEEPSTAKES FOR A CHANCE TO WIN A NEW TITANIUM!



wherein a complete set of features is developed outside of our main development branch to avoid disrupting other engineers. This development effort may go on for months, and affect hundreds or thousands of files. At the end of all of this, we need to get the changes back into our main branch intact. Without discussing too many details, I'll cover the process you would use with Perforce in this situation.

First, you'd tell Perforce to "integrate" changes from one branch to another. In other words, you are specifying what set of changes to merge without telling it specifically how you want that accomplished. You can limit the scope of the integration based on the path to the files, or specific versions of the files, but you can just as easily tell it to do the whole set of changes at once. Perforce will then "check-out" all of the files in the destination branch that were changed in the source branch. All of these files now have the special icon that indicates Perforce is waiting for you to specify how to resolve differences.

You would then tell Perforce to automatically resolve all the files that had no conflicting changes. This operation usually eliminates about 90%-95% of all changes with no manual work on the part of the engineer doing the merge. It can be done in a single step no matter how many files are involved.

Finally, you are left with a much smaller set of files that still have the special icon that indicates they have differences which have not yet been successfully resolved. Now you'll need to resort to interactively merging the conflicting differences using the visual three way merge tool provided by Perforce.

The above discussion may be too detailed for some, but the overall concept is that even when manipulating large sets of files, Perforce always tries to avoid involving you if it's not necessary, but if there are situations that need your attention, you will be involved, and given the detailed information you need to proceed efficiently. Perforce provides a consistent set of integrate & resolve functionality that is applied the same to all merging operations. It stands far ahead of CVS or SourceSafe in this respect, and ahead of most non-MacOS version control tools as well.

### INTER-FILE BRANCHING

Since we have been talking about merging between branches, I'll discuss the branching model used by Perforce. They call it Inter-File Branching, but in essence it is the use of the depot directory hierarchy to represent different branches of your development projects. The path to each individual file in the depot includes a full human readable representation of the intended purpose of that file. For example, it's easy to differentiate the intent of these two files:

```
//depot/Engineering/VectorWorks/ReleaseBranches/VectorWorks10.0
/AppSource/Project Setup.txt
//depot/Engineering/VectorWorks/TaskBranches/VW10/3DDDevelopme
nt/AppSource/Project Setup.txt
```

Behind the scenes supporting the seemingly simple Inter-File Branching concept is the Perforce database, which is aware of all branching relationships between any two files in the system. For every pair of files that has a branching relationship, it tracks the specific revisions that have been integrated. For large hierarchies of files that are related to other branched files, it's quite easy to display all changes made to a particular branch chronologically, or even to display all revisions in one branch that have yet to be merged to another.


To a large extent, the Inter-File Branching model works in concert with the automated merging capabilities described above. It allows independent branches within the codebase (and their independent change histories) to exist in a logical environment where every engineer cannot help but know how they are differentiated from each other simply by virtue of the path to the files. Inter-File Branching provides the conceptual foundation that can keep large teams of developers efficiently working on parallel development branches with very little management overhead.

Other tools such as CVS have a file hierarchy for your files, but then each file has a tree of numeric versions with no immediately apparent meaning. Thus every important event in CVS needs to be represented by a label that pulls together an arbitrary set of files and versions into a meaningful package. What an individual engineer needs to do in CVS to accomplish a simple task such as "Merge all of your version 4.1 changes into the main development branch" becomes so complex and error-prone that it prevents projects from even attempting to do large-scale parallel development. (Perforce also has labels, but they

high quality - competitive rates - 16 years experience - award winning

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

**Full Spectrum Software**  
**Development & Testing**



Device Drivers  
Carbon      Cocoa  
Real Basic  
Rescue Missions  
Cross Platform Development

**1661 Worcester Road**  
**Framingham, MA 01701**  
**508-620-6400**  
**www.FullSpectrumSoftware.com**

competitive rates - 16 years experience - award winning - high quality

22

PERFORCE

MACTECH • APRIL 2004



are rarely used because other Perforce functionality makes them much less necessary.)

SourceSafe has no meaningful tools to assist merging changes between branches and cannot effectively be used for any significant parallel development efforts.

### ATOMIC CHANGE SUBMISSION

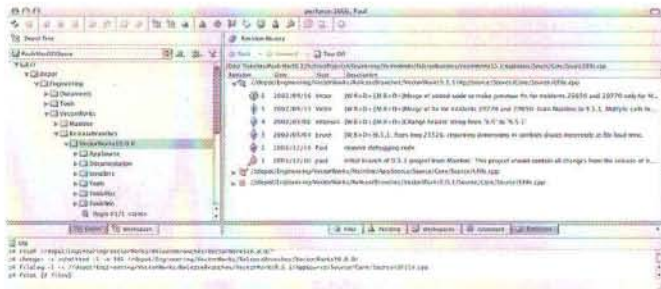


Figure 2 – Revision history of a single file with changelist comments

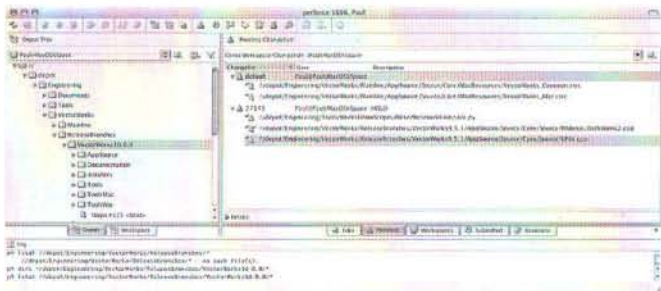


Figure 3 – Pending changelists before submission

Atomic change submission is another refinement that stems from the rigorous database architecture developed by Perforce at the outset. The concept is simple to grasp, and prevents a host of ugly side-effects which afflict competing products without this feature. Simply stated, if I try to submit changes to a set of files, and for whatever reason I am unable to change one or more files, then the entire submission is rejected. This dramatically improves the chances that the project as submitted to the version control system will be in a consistent, and buildable state, and improves the ability to analyze complex changes that have gone into the project after the fact.

In SourceSafe or CVS, if I submit ten files, and the last one has a conflict with a change that was already submitted by another engineer, I won't know it until the first nine have already been submitted. At that point, I may have a big problem, and will need to scramble to come up with a fix. If other engineers step into that trap, and check-in more files before the problems are solved, then the mess keeps getting bigger. In Perforce, if I submit ten files, and one of them has a conflict, then the entire submit fails before the central database is modified at all. I can then resolve the conflict without the pressure of having just checked-in a partial set of files which do not build.

In normal use, the organization of work allowed by Perforce changelists is also very beneficial. All of an engineer's open files are assigned to one or more pending changelists visible to all users of the system. (See Figure 3) The choice of which files to include as well as the description of the changes can all be prepared in advance to eliminate last-minute errors when submitting changes to the depot.

Unlike some other revision control systems where atomic change submission is tacked on as an afterthought, it is core to the implementation of Perforce. Every change that has been successfully submitted to the system is represented by the set of files that changed, a high level description of the significance of that change, and a list of files that were affected. The description of a change applies to the entire change (and all files that make up the change) rather than each individual file being given a duplicate of the description. The history of a hierarchy of files includes a list of the high level changes and their descriptions, not the less useful list of every file that changed between two dates. One can even implement server-side trigger scripts that can examine a proposed submission and programmatically accept or reject it in its entirety based on a centrally maintained submission policy.

### LOW ADMINISTRATION OVERHEAD

Perforce requires very, very little administration attention. If you install the server properly, set up your backups, and maintain the server hardware with adequate RAM and drive space, the only administrative attention required is upgrading the software as frequently or infrequently as you like, and a small amount of overhead when adding new user accounts or cleaning up after users who depart. Perforce has never once created an emergency for us, and I can't see a site needing a full time administrator for this system until it has many hundreds of users. Even then, I think there would be a lot of free time on that person's hands.

SourceSafe will quickly burden an administrator with unpleasant tasks such as the investigation and patching of file and database corruptions. Based on our experience, and that of others I've talked to, this is virtually certain over the long term with more than a handful of engineers using the system.

CVS will probably be less troublesome with regular maintenance, but the lack of thorough documentation and support and the need to assemble client and server pieces from various open source projects can certainly add to the initial outlay of effort.

### OTHER INFO

There are some other features that do not warrant extensive discussion, but are important to mention. Perforce maintains the ability to fully access all of the client functionality through their command line tools. This is the universal Perforce interface. It's available everywhere and can do anything that can be done with Perforce. This means that there is a backup plan if you ever run into something that can't easily be done using the GUI. It also



means that the system is highly scriptable and extensible, as you'd expect any mission-critical developer tool to be. It works well with most any scripting environment (Python, Perl, and possibly Ruby being the most commonly used).

### MAC SOFTWARE SUPPORT

Having talked at some length about the core system functionality, I'll spend some time talking about Mac-specific support. As far as the server goes, MacOS X is a supported platform as well as Linux & Windows. We use a Windows 2000 server platform, and there are no issues I'm aware of related to mixing Mac & Windows clients and servers.

The client software on the Mac includes MacOS X native command line tools as well as P4V - the visual GUI client, P4Web - a web browser based client, and a CodeWarrior plug-in. There are also legacy clients for MPW, older versions of CodeWarrior, and a MacOS 9 based version of P4Web. Finally, Apple has integrated native Perforce support into the Xcode development environment.

### GUI INTERFACE - P4WEB & P4V

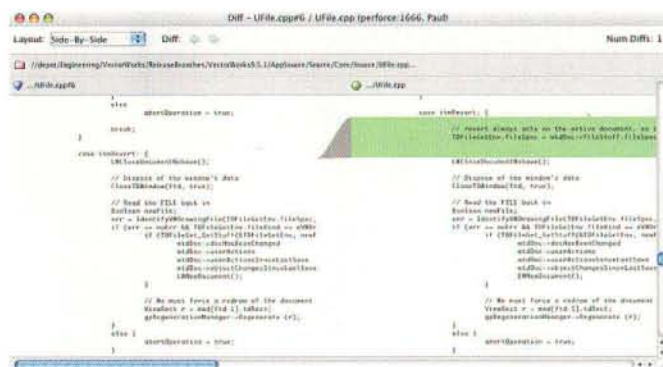


Figure 4 - Graphical text differences display

P4Web used to be the only "GUI" interface to Perforce on the Mac, and we used it successfully for a couple of years. It takes a little getting used to, but became quite easy to use, if a little slow. From my point of view, there is no reason for most users to continue using P4Web as the current version of P4V is faster, easier, more capable, and nicer looking. It introduces much better file differencing and merging capabilities, which were sorely lacking on the Mac under P4Web. Figure 4 is an example of the kind of text difference display P4V produces. You can easily display the difference between your local copy and any version in Perforce - or between any two versions of any file in the depot.

P4V is being developed by Perforce as the next generation GUI for Mac and Linux. With only a few exceptions, P4V has the full functionality of P4Win - the native Windows GUI client. In some respects, such as the graphical diff viewer, it's better. I think Perforce would ultimately like for P4V to be the only GUI client - even on Windows. They may have a way to go to

achieve that, but they are rapidly improving P4V, and it's already at a state of good usability.

### CODEWARRIOR INTEGRATION

The Perforce CodeWarrior plug-in is most useful for environments where CodeWarrior is the development platform, and you want quick access to syncing, checking out, and differencing files from within the IDE. You can submit changes from within the IDE, but the ability to resolve conflicts if they occur is weaker than either P4Web or P4V, and you'll probably gravitate to those more capable tools.

### XCODE INTEGRATION

Apple has integrated Perforce support directly into Xcode, and it provides the same feature set as their CVS integration - namely sync, check-out, check-in, and diff. As I don't use Xcode for production work, I'm not sure whether this integration works better than the CodeWarrior integration when submitting conflicting files. Either way, we don't find it difficult to use the GUI tools for the more demanding tasks.

### REFERENCES

There is a long list of major software companies, and projects with hundreds or thousands of developers who are happy Perforce users. See the customer spotlight page at <http://www.perforce.com/perforce/customers.html> for a bunch of interesting reading. Companies like Palm, Symantec, Macromedia, and TiVo, among many other household names, have standardized on Perforce as a best-of-breed solution for Mac development as well as virtually any other platform.

### PRICING

Current pricing for Perforce is \$750 per user, which includes a year of upgrades and support. Continuing the upgrade & support contract costs \$150 per user per year. Perforce has special site licensing available for educational institutions, free licenses available for open source projects, and an unlimited time evaluation version which includes two users and two workspaces. If you'd like to evaluate more in depth in a production environment, Perforce will supply you with a time-limited license enabling a larger number of seats, depending on your environment.

### FINAL WORD

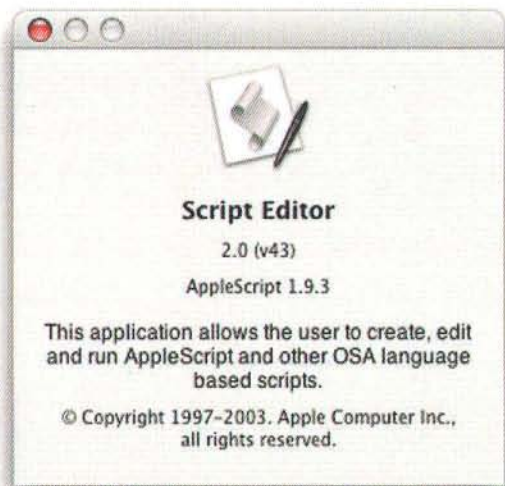
Perforce is a full-featured revision control system that differentiates itself from the competition by the uncompromising quality of its implementation. Mac support was acceptable three years ago, but due to recent improvements such as P4V for MacOS X, it is now very good, and still improving rapidly. Perforce is developed by a team that sets priorities early, and sticks to them. Producing a top-notch Mac development product is clearly one of their priorities. Oh - and if you're interested - Perforce is clearly one of the top players in Windows and Unix version control as well.



Copyright 2004 by Benjamin S. Waldie

# The “NEW” Script Editor

A few months ago, Dave Mark wrote a couple of articles about AppleScript. These articles touched briefly on topics including setting up the script menu in Mac OS X Panther, performing some basic Finder scripting, and accessing application dictionaries. This aims to be the first in a series of articles geared toward providing a more in-depth look at AppleScript.



**Figure 1.** The Script Menu's About Screen

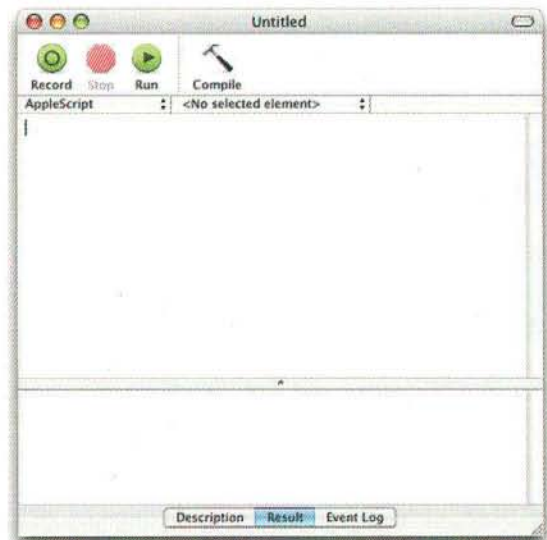
## PRIMARY INTERFACE CHANGES

This month's article starts at the beginning, and focuses on the new features in the new version of the Script Editor application, available with Mac OS X Panther (10.3). Those familiar with the Script Editor know that it has had some limitations over the years. For example, the Script Editor never had a find and replace, it had a limit on the number of characters that could be entered into a script document, and it was, of all things, not AppleScriptable. All of this has changed with the release of Script Editor 2.0!

## The Main Document Interface

The first thing that you will notice when you launch the new Script Editor, is that the interface looks different than that of previous versions. Since the entire application has been rebuilt from the ground up as a native Mac OS X application, it now sports this new look and feel.

The Script Editor is located in the `/Applications/AppleScript/` folder on your hard drive. Several other items exist in this folder as well, including a folder filled with fully editable example scripts.



**Figure 2.** The Script Editor's Document Window

First, let's take a look at the document window. You will notice that the top of each document window contains a toolbar. By default, this toolbar contains the expected buttons, including “Record”, “Stop”, “Run”, and “Compile” buttons. Like many other Mac OS X applications, users now have the ability to

Benjamin Waldie is president of Automated Workflows, LLC, a firm specializing in AppleScript and workflow automation consulting. In addition to his role as a consultant, Benjamin is an evangelist of AppleScript, and can frequently be seen presenting at Macintosh User Groups, Seybold Seminars, and MacWorld. For additional information about Benjamin, please visit <http://www.automatedworkflows.com>, or email Benjamin at [applescriptguru@mac.com](mailto:applescriptguru@mac.com).



customize the toolbar and add some other handy buttons for quick access. This can be done by selecting *Customize Toolbar* from the *View* menu.

The next thing that you will notice immediately is that the *Description*, *Result*, and *Event Log* are now located at the bottom of the document window. By clicking a tab, or by typing *command + 1, 2, or 3*, you can quickly and easily cycle between these views. If you prefer to have more space for editing your script, simply double click the window divider, or drag it to the desired height, to reduce the size of the tabbed area.

As in previous versions of the Script Editor, new document windows open to a predefined size. The default size of a new document window is still configurable, although Apple has moved this configuration to a new location. To set the default size of a window, first scale a window to the desired size. Next, select *Save as Default* from the *Window* menu.

### The Navigation Bar

Another new feature, which you may not notice right away, is a navigation bar, located directly above the code area in a script document window. If you don't see this navigation bar, select *Show Navigation Bar* from the *View* menu in order to make it visible.



Figure 3. The Navigation Bar

The navigation bar contains two popup menus. The first popup menu displays the current scripting language. The second popup menu displays a list of all of the elements in your script, including properties, globals, handlers, and script objects. By selecting an element in this popup list, the Script Editor will automatically navigate to that element and select it for you. In a script containing a large number of handlers, this can prove extremely useful. Unfortunately, there is not currently a way to sort this list of elements alphabetically. Rather, the elements are sorted in the order that they appear in the script.

A *handler*, also referred to as a *subroutine*, is a modular "chunk" of code that you write to perform one or more specific tasks. Once written, a handler may be called from multiple places in your script to perform the desired process. Handlers help to eliminate repetition in your code. By breaking your code up into smaller pieces in this manner, it also helps to make your code more organized, easier to debug, and easier to reuse in future scripts. We will review handlers in more detail in a future article.

### Result History and Event Log History Windows

The *Event Log* and *Result* tabs at the bottom of each document window display only the result or event log for the last time the current script was run. However, developers may need to retain this information for testing purposes.

In order to accommodate this need, the new Script Editor now contains an *Event Log History* window and a *Result History* window. These windows may be made visible by selecting them in the *Windows* menu. In order to function, they may also need to be enabled under *History* in the Script Editor's *Preferences*. There are also other options available under *History* in *Preferences* as well, including the number of results and event log entries to be stored.

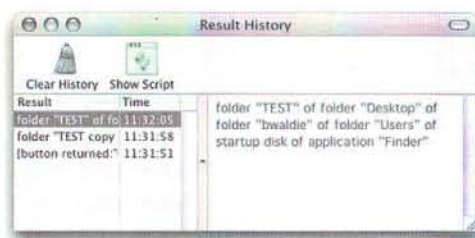


Figure 4. The Result History Window

The *Event Log History* window will store and display the event logs and the *Result History* window will store and display the results from the previous executions of any scripts that have been run. In addition, the script code that produced that result or event log is automatically stored and may be displayed at any time by double clicking on an entry in one of these windows, or by clicking the *Show Script* button in the window's toolbar. If the code has been modified since the entry was created, a new Script Editor window will be opened, which will contain the original code that was used to generate the entry!

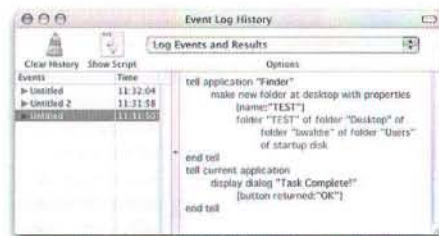


Figure 5. The Event Log History Window



The *Event Log History* and *Result History* windows are reset each time that the Script Editor is launched.

### Library Palette

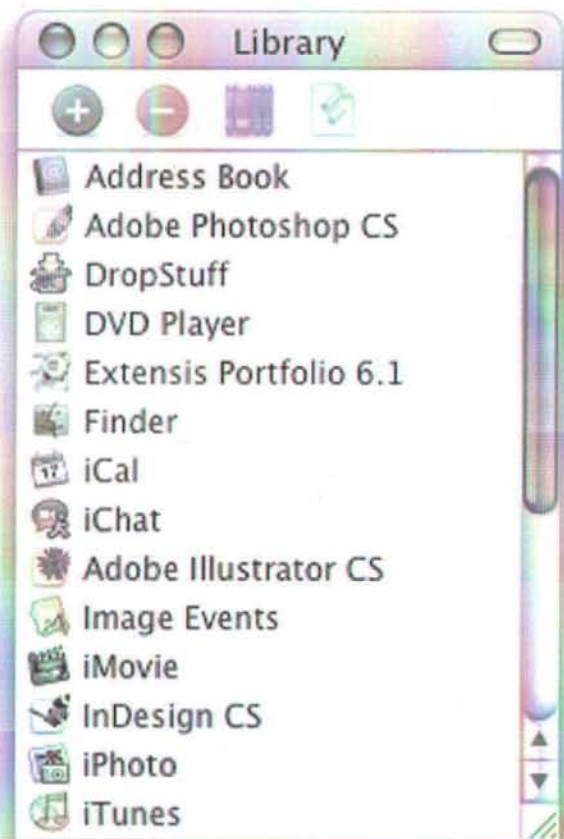


Figure 6. The Library Palette

In past versions of Mac OS X, opening application and scripting addition dictionaries in the Script Editor sometimes proved to be a time consuming process. First, you had to select *Open Dictionary* from the File menu. Then, after a delay, a window would prompt you to select a scriptable application on your machine. The ability to open dictionaries in this manner still exists in the new Script Editor. However, those who prefer an easier way may want to check out the new *Library* palette, which can be displayed by selecting *Library* from the *Window* menu.

By default, the *Library* palette displays a list of several commonly accessed scriptable applications and scripting additions installed on your machine. Additional applications and scripting additions may be manually added to the list by dragging and dropping them from the Finder directly into the palette. You may also add them by clicking the + button in the window's toolbar, which you will prompt you to select an application or scripting addition.

To quickly open the dictionary for an application in the list, double click on it, or click on the dictionary icon in the window's toolbar.

Clicking the script icon in the toolbar of the *Library* palette will create a new Script Editor document containing a tell block for the selected application. For example, clicking the script icon after selecting the Finder would generate a new window containing the following code:

```
tell application "Finder"
end tell
```

### New Dictionary Layout

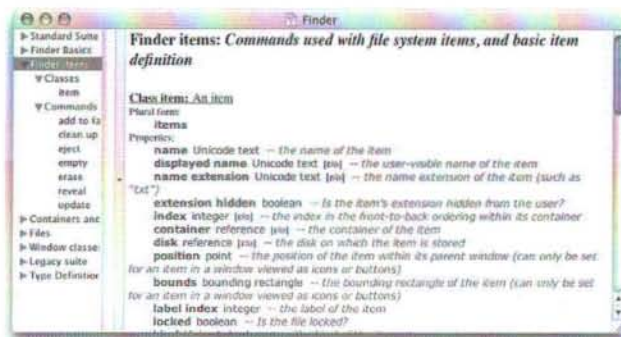


Figure 7. The Finder Dictionary Window

When opening application or scripting addition dictionaries, you will notice some changes as well. Primarily, the list of application terminology on the left side of the window is now organized into groups, also referred to as *suites*. Each group may be expanded to display collapsible lists of *classes*, or objects and properties, and *commands*.

### NEW EDITING FEATURES

Let's begin looking at some of the new script editing features of the Script Editor, which are designed to save you time and help you to develop your scripts more efficiently.

#### Find And Replace

Finally, at long last, probably one of the most requested features of all time has been added to the Script Editor application – the all important *Find and Replace*! To display the find and replace window, type *command + F*, or select *Find* from the *Edit > Find* menu. Once you begin using this feature, you will wonder how you ever worked without it. It is a tremendous time saver when writing code. In addition, dictionary windows are also now fully searchable, which proves to be quite valuable when looking for a specific term.



**EXTREME<sup>to</sup> the MACS!**

**Aria Extreme**

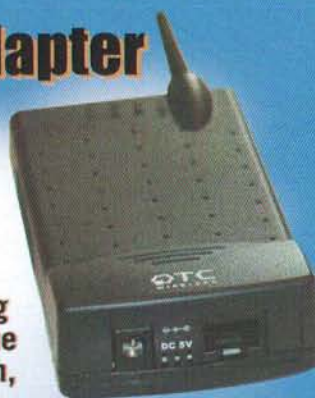
**\$78.95**



Add AirPort Extreme to any G3 or G4 PowerBook! Get blazing speeds up to 54 Mbps (five times AirPort Speed!), comes with an external stub antenna for better range than internal cards! (requires Mac OS X 10.2.6 or above)

**OTC Wireless  
802.11g Ethernet Adapter**

**\$164.95**



Driverless Ethernet to 802.11g converter. Simply plug into the Ethernet port of any Macintosh, PC, or even printer to put it transparently on your 802.11g (54 Mbps) network! Use with a hub to enable multiple devices with one adapter!

**DEV DEPOT<sup>®</sup>**

[www.devdepot.com](http://www.devdepot.com)

**877-DEPOT-NOW**

**Use AirPort  
Extreme with  
any Macintosh!**

Toll Free: 877-337-6866 • Outside US/Canada: 805-494-9797 • Fax: 805-494-9798 • [orders@devdepot.com](mailto:orders@devdepot.com)  
PO Box 5200 • Westlake Village, CA 91359-5200



## AirPort Extreme (802.11g) PCI Card

**\$98.95**



Easy to install! Step up to  
AirPort Extreme (54 Mbps)  
for any PCI Macintosh!  
(requires Mac OS X 10.2.6 or above)

AirPort Extreme

## AirPort Extreme Omni Antenna 5 dBi

**\$78.95**



Want more range from your AirPort  
Extreme base station? This 5 decibel  
antenna is 30% more powerful than  
other brands!

**We also offer a full line of AirPort (802.11b) products!**

**Mac Wireless  
USB  
to 802.11b  
Converter**



**\$78.95**

**AirPort PCI  
to 802.11b  
Converter**



**\$128.95**

**AirPort 802.11b  
PCMCIA Card**



**\$88.95**

**Belkin Wireless  
Cable/DSL Gateway  
Router**



**\$112.95**



## Script Assistant

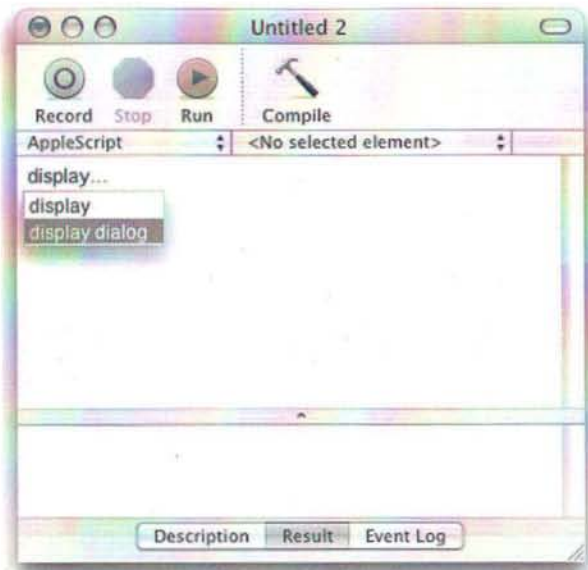


Figure 8. The Script Assistant

If you have used Xcode, or other professional development environments in the past, you may be familiar with the term “code completion.” This is another great feature that has been built into the new Script Editor.

In order to enable code completion, select *Preferences* from the *Script Editor* menu, and click on the *Editing* icon in the preference window toolbar. Next, select the *Use Script Assistant* checkbox. You will need to re-launch the Script Editor for the change to take effect.

Once enabled, the script assistant monitors your code as you type and attempts to help you auto-complete the code whenever possible. You will know when code completion is possible because you will see three dots (...) appear after the text you are typing. To auto complete the code, press the *F5* key. This will display a list of the suggested AppleScript terms to use. To select a term, use the up/down arrow keys to select the desired term, and then press *Enter*.

### New and Exciting Save Options

In previous versions of the Script Editor, there were three main ways to save your scripts – as *Applications*, as *Compiled Scripts*, or in *Text Format*. In addition to allowing you to save in these familiar formats, the new Script Editor also offers two new options – *Script Bundles* and *Application Bundles*!

It is important to note that scripts saved as bundles will not function on systems prior to Mac OS X Panther (10.3). When saving as a bundle, your script is actually saved as a folder, which has the appearance of a single file to the user. A saved bundle contains sub-folders and all resources of your script.



Figure 9. A Script Bundle's Folder Structure

To view and edit the resources of a bundled script, *control click* on the script in the Finder, and select *Show Package Contents* from the contextual menu that is displayed. If desired, you may add additional resources to the bundle, including images, icons, script libraries, and more.

Bundles will even allow you to embed scripting additions directly into your script, eliminating the need to install those scripting additions on machines that will run the script. To embed a scripting addition into a script bundle, create a folder named *Scripting Additions* in the *Contents > Resources* folder in the bundle, and install the scripting addition into this folder. Please note that the embedded scripting addition must actually be installed on the machine that creates the script bundle.

### AppleScript Support

Ironically, older versions of the Script Editor lacked an important feature, AppleScript support. In other words, they weren't scriptable. The new Script Editor is now fully scriptable, allowing you to begin automating even the creation of your scripts with AppleScript.



Figure 10. Contextual Menus



In addition to being scriptable, the new Script Editor is also attachable, and allows you to trigger scripts from within a contextual menu. To view the Script Editor's contextual menu, *control click* anywhere in the code of your script. You will notice that Apple has included quite a number of pre-built scripts for you to use, which can do things like add error protection around the selected code in the current Script Editor document.

If desired, you can also add your own scripts into the contextual menu. To do so, select "Open Scripts Folder" from the contextual menu, and add your script(s) into the folder structure. In addition to being displayed in the contextual menu, the scripts in this folder are also displayed in the system-wide *Script Menu*, which may be enabled by launching the *Install Script Menu* application, located in the *Applications > AppleScript* folder.

Since the Script Editor is now scriptable, Apple has also added some interesting new AppleScript features into the *Services* menu. The *Services* menu can typically be found in the main menu of most applications in Mac OS X. New AppleScript features in the *Services* menu include the ability to select text in a document and...

- Run it as an AppleScript
- Open a new Script Editor document containing the text
- Run it as an AppleScript and replace the selected text with the result

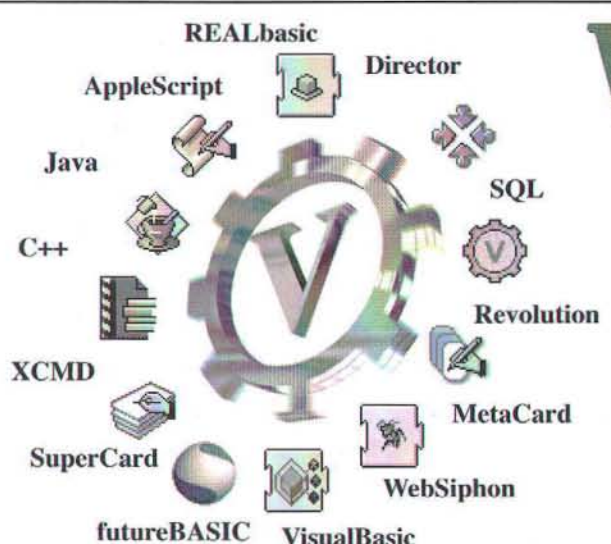
For those who are interested in sharing their code on the Internet, Apple has also introduced URL Protocol support into the new script editor. In other words, users can embed AppleScript code in web pages as links. Then, clicking those links will cause the Script Editor to perform a specific action with the embedded AppleScript code. Links can be configured to:

- Create a new Script Editor document containing the embedded AppleScript code
- Insert the embedded AppleScript code at the current position in the front Script Editor document
- Add the embedded AppleScript code to the end of the front Script Editor document

#### IN CLOSING

We have explored the primary new features of the updated Script Editor in Mac OS X Panther (10.3). For additional information about the new Script Editor application, please visit the Script Editor page on Apple's AppleScript web site at <http://www.apple.com/applescript/scripteditor/>.

There are still many other new and exciting AppleScript changes in Mac OS X Panther to explore. In fact, the release of Panther brought with it probably the largest number of AppleScript features and changes since AppleScript was first introduced. In the coming months, we will touch on some of these new AppleScript-related features in Mac OS X. In addition, I will begin to provide you with information designed to teach you the basics of AppleScripting, so that you can put yourself on the way to becoming an expert scripter. Until next time, keep scripting!



# Valentina

## Object-Relational SQL Database

### The fastest database engine for MacOS/Windows

**It operates 100's and sometimes  
a 1000 times faster than other systems**

**[www.paradigmasoft.com](http://www.paradigmasoft.com)**

Hosted by [macserve.net](http://macserve.net)  
Download full featured evaluation version



by Tim Monroe

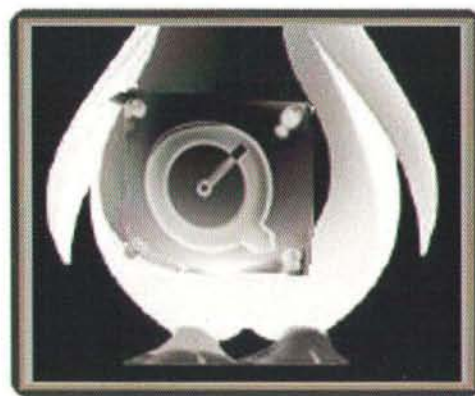
# Snow Day

## *Developing QuickTime Applications for the iPod*

### INTRODUCTION

It's commonly believed that Apple's iPod portable music player uses a bare-bones operating system developed by a company called Pixo, Inc., and that the device itself is a closed system insofar as only Apple can add software modules — such as the games or contacts list manager — to the device. And, it's further believed, even if there *were* a way to load third-party applications onto the device, actually writing such applications requires access to development tools whose availability is tightly controlled and that are expensive to license. In effect, the iPod is even less open than the original Macintosh computer (which is to say, not very open at all).

In fact, none of these beliefs is true. Once we know what's going on, it's fairly simple to develop quite powerful applications for the iPod using nothing more than a text editor and a programming language that most of us had considered to be long since obsolete. In this article, I'll show just how easy it is to write iPod applications by developing an application that can open and display QuickTime movies on the iPod. In addition, I'll show how to support the standard editing operations (cut, copy, paste, and undo). The result will be an iPod-based version of our QuickTime sample application, QTShell. For reasons that will become clear soon, let's call this application "SnoEez". **Figure 1** shows a frame of a movie being displayed by SnoEez. (You may recognize this as a cropped version of our standard penguin movie, with the X-ray video effect applied to the entire movie.)



**Figure 1:** A movie being played by SnoEez

We'll begin by figuring out which operating system really powers the iPod and which tools we need to use to develop custom applications for it. Then we'll jump into developing SnoEez.

### SOME DETECTIVE WORK

It's easy enough to cast doubt on the idea that the iPod runs an operating system developed by Pixo (which is not, of course, to be confused with Pixar Animation Studios). Pixo is a small company that indeed makes an OS for portable devices like cell phones and PDAs. But it was recently acquired by Sun Microsystems, and it's difficult to imagine that Apple would allow a key piece of technology for one of its most popular and most lucrative products to fall into the hands of a competitor. As we'll see, the iPod does rely on some Pixo software, but that software is just a thin layer of imaging software, not the entire operating system. The Pixo software handles screen drawing and item selection, but not much more. In a nutshell, Pixo's software is to the iPod as QuickDraw was to the original Macintosh: a key element, perhaps, but not nearly the whole enchilada. And, more importantly, the Pixo software in the iPod is likely to be phased out in the future, just as QuickDraw is being phased out in favor of Mac OS X's Quartz imaging layer.

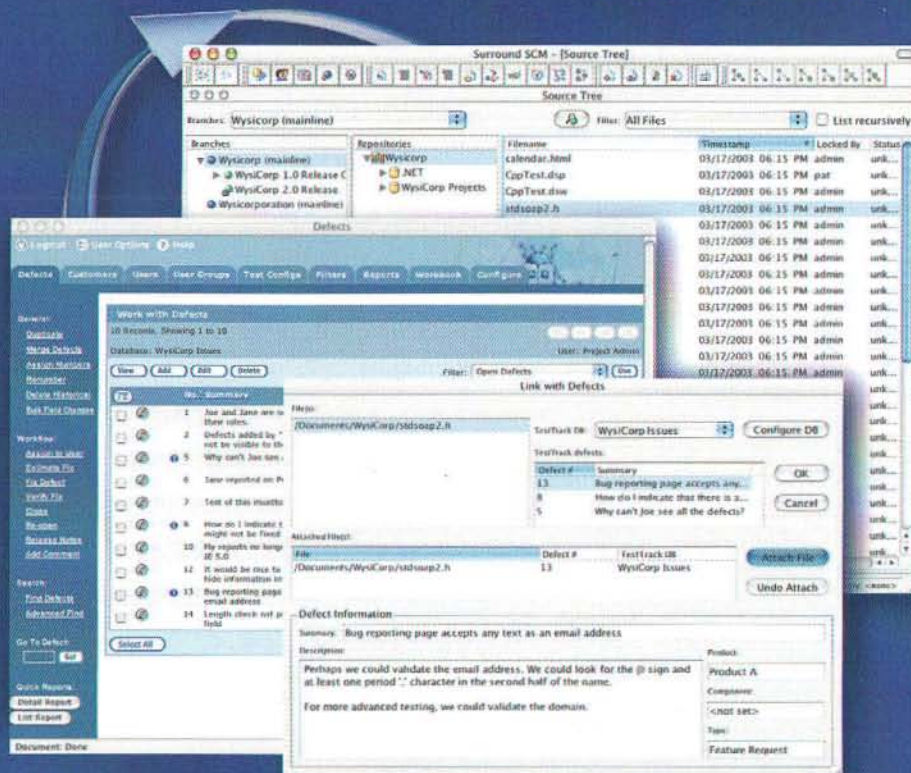
**Tim Monroe** is a member of the QuickTime engineering team at Apple. You can contact him at [monroe@mactech.com](mailto:monroe@mactech.com). The views expressed here are not necessarily shared by his employer.



# Complete Source Control and Defect Management

Seapine Software™  
changing the world  
of software development

## for Mac OS X



**Effective source code control and defect tracking require powerful, flexible, and easy-to-use tools—Surround SCM and TestTrack Pro**

- Complete source code control with private workspaces, automatic merging, role-based security, and more
- Comprehensive defect management — track bug reports and change requests, define workflow, customize fields
- Fast and secure remote access to your source files and defects — work from anywhere
- Advanced branching simplifies managing multiple versions of your products
- Link code changes with defects and change requests — know who changed what, when, and why
- Scalable and reliable cross-platform, client/server solutions support Mac OS X, Windows, Linux, and Solaris
- Exchange data using XML and ODBC, extend and automate with SOAP support
- Licenses priced to fit your budget

**Seapine Software Product Lifecycle Management**  
**Award winning, easy-to-use software development tools**

Seapine  
**Surround SCM**  
Seapine  
**TestTrack PRO**



**Download Surround SCM  
and TestTrack Pro at  
[www.seapine.com](http://www.seapine.com)  
or call 1-888-683-6456**

all product names listed herein are registered trademarks of their respective owners. All rights reserved.





## Finding the Real Operating System

Let's begin by doing a little detective work to figure out what operating system really is powering the iPod. Connect your iPod to your Mac OS X computer and then, in a Terminal window, navigate to the iPod. For instance, if your iPod is named, say, "iDegger", navigate to `/Volumes/iDegger`. In this directory there is a folder called `iPod_Control`. List its contents; you'll see something like this:

```
[Kant:/Volumes/iDegger/iPod_Control] monroe% ls -l
total 8
drwxrwxrwx  4 monroe  unknown  136 Mar  4 13:55 Device
drwxrwxrwx 22 monroe  unknown  748 Feb 26 2002 Music
-rwxr-xr-x  1 monroe  unknown   38 Mar  6 14:23 iPodPrefs
drwxrwxrwx 12 monroe  unknown  408 Mar  6 14:23 iTunes
```

For present purposes, we want to look at the `Device` folder. Listing its contents gives us this:

```
[Kant:/Volumes/iDegger/iPod_Control/Device] monroe% ls -l
total 4
-rwxrwxrwx  1 monroe  unknown  2872 Mar  4 13:55 Prefs
-rwxrwxrwx  1 monroe  unknown   413 Mar  4 05:55 SysInfo
```

That's not very revealing. But if we list the contents of the `Device` folder using `ls` with the undocumented option `af` (for "all files"?), we get a far different picture:

```
[Kant:/Volumes/iDegger/iPod_Control/Device] monroe% ls -laf
total 41
-rwxrwxrwx  1 monroe  unknown  2872 Mar  4 13:55 Prefs
-rwxrwxrwx  1 monroe  unknown   413 Mar  4 05:55 SysInfo
drwxr-xr-x  5 root    wheel    190 Sep 22 08:33 bin
drwxrwxrwt  6 root    wheel    204 Jan 29 2003 cores
dr-xr-xr-x  2 root    wheel    512 Mar  6 13:38 dev
lrwxrwxr-t  1 root    admin    11 Mar  6 13:38 etc
-r--r--r-  1 root    admin  709440 Mar  6 13:38 mach.sym
-rw-r--r-  1 root    wheel  744576 Apr  1 15:21 mach_krn
```

That's right, the iPod is running a slimmed-down version of Mac OS X! Why am I not surprised?

## Finding the Executable Format

So far, so good. At this point it might seem like we're done, since all we need to do is compile our application using the standard development tools like Xcode and Interface Builder and then move it into some appropriate location in the iPod. After all, if the iPod is really running Mac OS X, then the executable format is well known (Mach-O).

But of course it's not that simple. The main problem is that the iPod does not have the graphics horsepower to support the Quartz imaging system. That's where the Pixo software comes into play. Pixo provides the imaging system for the iPod, which is a collection of *screens* of information. Some of these screens present lists, such as the "menu" screens or the lists of songs or playlists; some of these screens present linear controls managed with the iPod's scroll wheel; and some of these screens present raw data (as in **Figure 1**). So ultimately we need to figure out how to access the Pixo imaging primitives.

First, however, to continue our detective work, let's open one of the existing iPod applications with a hex editor. If we open, say, the Solitaire game, we'll see this stream of hex values:

```
37 4A 6F 79 21 70 65 66 66 70 77 70 63 60 5C 59
56 61 4e 56 5f 52 0a 37 60 5b 5c 57 5c 4f 2d 63
52 5f 60 56 5c 5b 2d 3e 3b 40 0a 37 35 50 36 2d
4e 5d 5d 59 52 2d 50 5c 5a 5d 62 61 52 5f 39 2d
56 5b 50 3b 0a 2d 2d 51 52 53 56 5b 52 35 34 56
5b 56 61 50 4e 5f 51 60 35 36 34 36 2d 47 56 5b
35 56 61 50 4e 5f 51 60 6c 52 5b 51 36 0a ...
```

In ASCII format, this looks like this:

```
7Joy!peffpwp^YVaNV_R.7^[W\O-c
R^V\[->@.75P6-N]]YR-P\Z]baR_9-
V[P;.-QRSV[R54V[VaPN_Q`5646-G5V
[VaPN_Q`IR[Q6.
```

The "Joy!peff" at the beginning of the data stream might lead us to believe that the executable format is not Mach-O, but the earlier PEF format adopted from IBM when the PowerPC chips were first used in Macintosh computers. But, once again, we would be wrong. If perchance we are clever enough to subtract 13 from each ASCII value, we get this stream of bytes (splitting the lines whenever we encounter a newline character):

```
*=bl.cXYcgcv SOLITAIRE
*SNOJOB VERSION 1.3
*(C) APPLE COMPUTER, INC.
  DEFINE('INITCARDS()') : (INITCARDS_END)
```

Lo and behold! We've cracked the riddle. The Solitaire game isn't a compiled application at all; rather, it's just a script of commands encoded using a version of the popular rot-13 algorithm. The script language is strongly reminiscent of a venerable programming language called SNOBOL. Judging by the second comment line (introduced with the "\*" character), it looks like Apple is using a previously unknown variant of the language called SNOJOB. Clever indeed.

## SNOBOL

In the interest of saving time, I'll spare you any more tedious reverse engineering and just present the results of my investigations. As best I can tell, SNOJOB is simply classic SNOBOL with bindings to the Pixo imaging system. The SNOBOL language was first developed in the early 1960's by a team of researchers at Bell Laboratories led by Ralph Griswold. The name is an acronym of sorts for "String-Oriented Symbolic Language", which reflects the focus of the language on processing strings and matching patterns. In fact, SNOBOL is one of the few languages ever devised in which patterns (or regular expressions) are full-fledged objects in the language.

We don't have space for a complete investigation of the SNOBOL language, so let's focus on a few salient features. Perhaps the most striking feature of the language is its utter simplicity. It provides no control structures as we would understand them nowadays, and no type declarations. A SNOBOL script is a series of lines, and each line has this structure:



*label*      *expression*      *branching-info*

The *label* is any arbitrary string that uniquely identifies the line; this label provides a means for other lines in the script (or indeed the same line) to branch to that line. The *expression* is any legal SNOBOL expression, which always returns a value. The *branching-info* is interpreted on the basis of that value. If the branching information is of the form “:S(*label*)”, then if the evaluation of the expression is successful, the program continues executing at the line with the specified label. If the branching information is of the form “:F(*label*)”, then if the evaluation of the expression is unsuccessful, the program continues executing at the line with the specified label. Finally, if the branching information is of the form “:(*label*)”, then the program continues executing at the line with the specified label whether the evaluation of the expression succeeds or fails.

Any of the three parts of a line of script can be omitted, and the branching information can include both :S and :F items. If a line contains no branching information, then the program continues executing at the next line. If present, a label must be the first item on the line, and the expression must be indented if no label is present.

The valid expressions include checks for equality like “EQ(N,10)” and calls to user-defined functions. They also include pattern matches, which are indicated by the space character “ ”. (Yes, white space is an operator in the SNOBOL language. No, I'm not making this up!) So the following expression branches to a line labeled “LN1” if the value of the variable OS\_TYPE matches the pattern “Darwin”:

```
OS_TYPE "Darwin"      :S(LN1)
```

To match the pattern “Darwin”, that value could be “Darwin” or “Darwinian” or the like.

**Listing 1** shows a SNOBOL version of the *basename* function we've encountered in several previous articles. Given a full pathname, it returns the portion of the pathname following the rightmost path separator.

#### Listing 1: Getting the basename of a filename

```
DEFINE('basename(fName)')      : (basename_END)
basename
    seg = fName
LP  seg BREAK('/') LEN(1) REM . seg :S(LP)
    basename = seg              : (RETURN)
basename_END
```

(It's perhaps useful to know that SNOBOL matches the *smallest* possible string.)

SNOBOL is distinctive among scripting languages in that the SNOBOL interpreter does *not* first parse a script and then process its input according to the parsed script (as is the case with more common scripting languages like Sed, Awk, or Perl.) Rather, the SNOBOL interpreter just executes each line in sequence, according to the evaluation and branching instructions specified in the script.

There is, to repeat, no separate parsing phase. That's why, as you can see, we need to branch unconditionally to the *basename\_END* label after executing the **DEFINE** command (which declares to the interpreter that “*basename*” is a user-defined function). Otherwise, the body of the function would be executed immediately, not just when the function is invoked by a line of script.

To read input, we need to make an explicit call to the **INPUT** command. **Listing 2** shows our script's “main event loop”, which reads input and processes it.

#### Listing 2: Handling commands

```
* process commands from the standard input
INP inputLine = INPUT      :F(END)

* match the input line
inputLine 'doOpen'        :S(OPN)
inputLine 'doNew'         :S(NEW)
inputLine 'doClose'       :S(CLS)
inputLine 'doSave'        :S(SAV)
inputLine 'doSaveAs'      :S(SVA)
inputLine 'doExit'        :S(EXT)
inputLine 'doUndo'        :S(UND)
inputLine 'doEdit' REM . OPERATION :S(EDT)
inputLine 'doSelect'      :S(SLT)
inputLine 'doAbout'       :S(ABT)
inputLine 'openMovie' REM . ARGS  :S(OPF)
inputLine 'doButton' REM . ARGS  :S(BTN)

* nothing matched; go get another line...
: (INP)

* handle the commands matched just above
OPN doOpen()              : (INP)
NEW doNew()               : (INP)
CLS doClose()             : (INP)
SAV doSave()              : (INP)
SVA doSaveAs()            : (INP)
EXT doExit()              : (INP)
UND doUndo()              : (INP)
EDT doEdit(OPERATION)     : (INP)
SLT doSelect()            : (INP)
ABT doAbout()             : (INP)

OPF ARGS BREAK(' ') . FILE LEN(1) REM . ISNEW
+ openMovie(FILE, ISNEW) : (INP)
BTN ARGS BREAK(' ') . WINDOW LEN(1) REM . BUT
+ doButton(WINDOW, BUT)  : (INP)
END
```

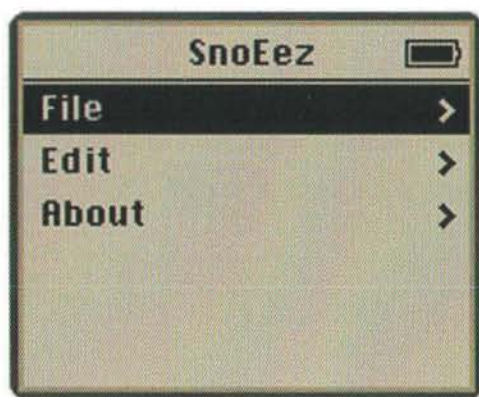
You'll notice that we read a line of input, match it against patterns we know how to process, and then branch unconditionally back to the **INP** label, which reads another line of input. Only when we encounter an error reading input or when some user-defined function branches to the **END** label does the script stop processing. (By the way, the “+” character as the first character on a line indicates a continuation line.)

But where does the input for SnoEez come from? As we'll see shortly, it's generated by SnoEez itself, when handling user actions. First however let's see how to construct our application's user interface.

#### SCREENS

An application running on the iPod consists of a series of *screens*. An application's *main screen* is the first screen displayed when the application is launched. **Figure 2** shows the main screen of SnoEez.



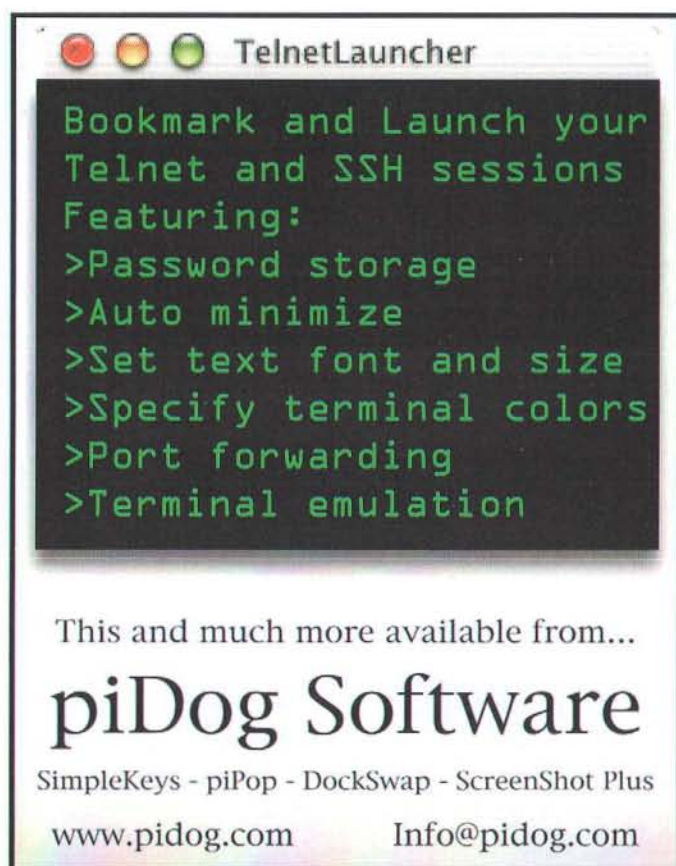


**Figure 2:** The main screen of SnoFez

This main screen is also a *menu screen*, meaning that it consists of a list of items that can be selected using the iPod's scroll wheel and selection button (the button in the middle of the scroll wheel). We can define a main screen using the Pico function `NEW_SCREEN` like this:

```
main = NEW SCREEN(kMenuType, "SnoEez", kMainScreen)
```

We add items to a menu screen using the `ADD_ITEM` function. SnoEcz' main screen is constructed using these lines of code:



```
ADD_ITEM(main, "File", kNoFunction, file)
ADD_ITEM(main, "Edit", kNoFunction, edit)
ADD_ITEM(main, "About", kNoFunction, help)
```

Here the fourth parameters are identifiers for *child screens* (or *subscreens*), which must have been defined before we call ADD ITEM. We can construct these subscreens like this:

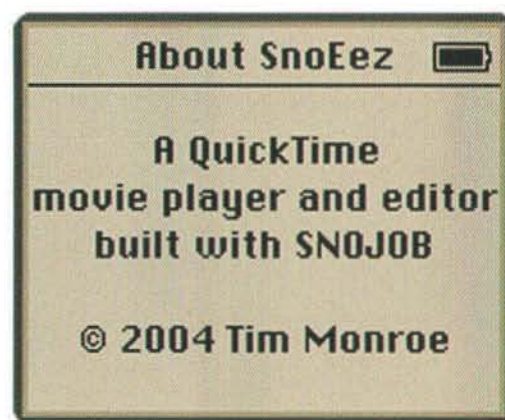
```
file = NEW_SCREEN(kMenuType, "SnoEez \u0401 File", main)
edit = NEW_SCREEN(kMenuType, "SnoEez \u0401 Edit", main)
help = NEW_SCREEN(kTextType, "About SnoEez", main)
```

As you might surmise, the fourth parameter here is the *parent screen*, which is the screen selected when the user presses the Menu button on the iPod.

Notice that the `help` screen is declared to be of type `kTextType`. This indicates that that screen consists solely of text. We can add text to the `help` screen with this code:

```
ADD_TEXT(help, "A QuickTime\nmovie player and  
+ editor\nbuilt with SNOJOB\n\n\n\\u169 2004 Tim Monroe")
```

This gives us the About screen shown in **Figure 3**. `ADD_TEXT` understands C-style escape sequences (such as `"\n"`) and Unicode escape sequences (such as `"\u169"` for the copyright symbol "©").



**Figure 3:** The About screen

It's easy enough to populate the File and Edit screens (**Figures 4 and 5**). The only new thing to learn is that we need to specify the name of the function to call, along with any parameters, when a screen item is selected. For instance, we can add a New item to the File screen like this:

```
ADD_ITEM(file, "New", "doNew", kNoChildScreen)
```

When the user selects the New item in the File screen, SNOJOB interpreter passes the string "doNew" back to the application as part of the standard input. As we saw in **Listing 2**, our script matches that string and calls the `doNew` function.



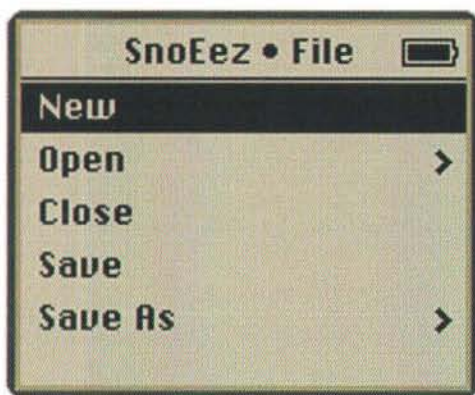


Figure 4: The File screen

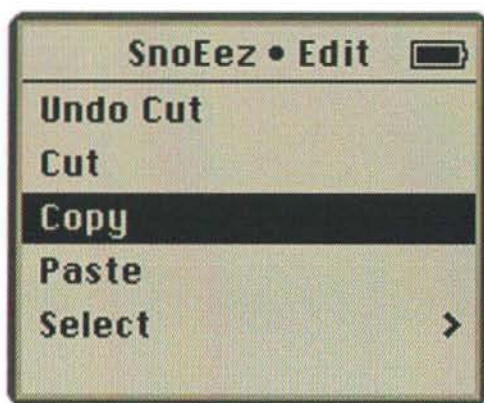


Figure 5: The Edit screen

Listing 3 shows the complete definition of the `setUpScreens` function.

#### Listing 3: Setting up the screens

```

DEFINE('setUpScreens()')           : (setUpScreens_END)
setUpScreens
*
  create a new main screen
  main = NEW_SCREEN(kMenuType, "SnoEez", kMainScreen)
  file = NEW_SCREEN(kMenuType, "SnoEez \u401 File", main)
  edit = NEW_SCREEN(kMenuType, "SnoEez \u401 Edit", main)
  help = NEW_SCREEN(kTextType, "About SnoEez", main)

  ADD_ITEM(main, "File", kNoFunction, file)
  ADD_ITEM(main, "Edit", kNoFunction, edit)
  ADD_ITEM(main, "About", kNoFunction, help)

**** File screen
  ADD_ITEM(file, "New", "doNew", kNoChildScreen)
  ADD_ITEM(file, "Open", "doOpen", kDeferredChild)
  ADD_ITEM(file, "Close", "doClose", kNoChildScreen)
  ADD_ITEM(file, "Save", "doSave", kNoChildScreen)
  ADD_ITEM(file, "Save As", "doSaveAs", kDeferredChild)

**** Edit screen
  ADD_ITEM(edit, "Undo", "doEdit undo", kNoChildScreen)
  ADD_ITEM(edit, "Cut", "doEdit cut", kNoChildScreen)
  ADD_ITEM(edit, "Copy", "doEdit copy", kNoChildScreen)
  ADD_ITEM(edit, "Paste", "doEdit paste", kNoChildScreen)
  ADD_ITEM(edit, "Select", "doSelect", kDeferredChild)

**** About screen

```

```

  ADD_TEXT(help, "A QuickTime\nmovie player and
+ editor\nbuilt with SNOJOB\n\n\u169 2004 Tim Monroe")
  : (RETURN)
setUpScreens_END

```

Several commands use the constant `kDeferredChild` as their fourth parameter. This allows us to pretend for the moment that the menu item has a child screen without having already created one explicitly. The function invoked by the item selection will need to create the child screen.

#### MOVIES

So how does QuickTime fit into this picture? SNOJOB has the following useful property: any call to a function that is not a built-in SNOBOL function (for instance, `LEN`), a Pixo user-interface function (for instance, `ADD_ITEM`), or a user-defined function (for instance, `setUpScreens`) is assumed to be contained in CarbonLib. The parameters to these external functions are specified in the script as strings but are coerced to the proper type by the SNOJOB interpreter. Opening and controlling QuickTime movies is therefore just a matter of resorting to the Carbon-based QuickTime APIs whenever necessary.

#### Opening a Movie File

Let's start by seeing how to open a movie file. As we've seen (in Listing 3), selecting the Open item in the File screen results in our script's `doOpen` function being called. The first thing we need to do is display a list of the QuickTime movies stored on the iPod, as shown in Figure 6.



Figure 6: The list of movies

The Pixo user-interface software provides an extremely easy way to do this:

```

fileName = NEW_LIST_SCREEN("Movies", "Movies", file)

```

Here the first parameter is the title of the new movie-selection screen. The second parameter is the name of the folder that contains the QuickTime movies we want to display. This folder is assumed to reside within the `iPod_Control` folder we encountered earlier. The third parameter is of course the parent screen.



If the user selects one of the movie files displayed in the movie list, then its name is returned and assigned to the `fileName` variable. If the user cancels the movie file selection by pressing the iPod's Menu button, an empty string is returned. We exit `doOpen` if `fileName` is the empty string with this code:

```
IDENT(fileName)                :S(RETURN)
```

(The `IDENT` function compares two or more strings for identity and succeeds if they are character-for-character identical; in any SNOBOL function, missing parameters are taken to be the empty string.)

We can open the selected file by executing the Carbon functions `OpenMovieFile` and `NewMovieFromFile`. The only "gotcha" is that `OpenMovieFile` called from a SNOJOB script is assumed to pass a pathname, not a file system specification.

```
OpenMovieFile(fileName, refNum, fsRdWrPerm)
NewMovieFromFile(movie, refNum, resID, 0,
+             newMovieActive)
```

Now we need to create a new screen within which the movie will be displayed. Once again we'll call `NEW_SCREEN`, this time like this:

```
moov = NEW_SCREEN(kDataType, "", main)
```

This indicates that the `moov` screen is a *data screen*, where the entire contents of the screen are supplied by the application.

As always, the preferred way to manage playback of QuickTime movies is by attaching a movie controller to the movie, which we can do with these lines of code:

```
SetRect(rect, 0, 0, ScreenWid(), ScreenHgt())
mc = NewMovieController(movie, rect, mcTopLeftMovie)
```

The functions `ScreenWid` and `ScreenHgt` are built-in Pixo functions that return the width and height of the entire iPod screen. By using these functions instead of hardcoding values, we allow our application to run on all present and future iPods.

To allow the user to edit movies, we need to explicitly enable editing:

```
MCEnableEditing(mc, 1)
```

Finally, we need to set the movie's graphics world to the data screen:

```
SetMovieGWorld(movie, moov)
```

The Pixo user-interface layer takes care of moving pixels drawn by the movie controller into the data screen `moov`.

### Keeping Track of Movie Information

Since `SnoEez` can open multiple movies and (for instance) allow the user to cut and paste data from one movie to another, we need to keep track of the movie controller and other pieces of data

associated with any particular movie. SNOBOL does not provide any mechanism for defining structured data types, but we can simulate such types with *associative arrays* (also known as *dictionaries* or *hashes*). So we'll maintain a global associative array called `appData`. In fact, we'll use the `appData` array for two purposes, to hold global data values and to hold data associated with a particular window. **Listing 4** shows the definition of the `initApp` function, which allocates the global array and initializes its values.

### Listing 4: Initializing the application

```
initApp
DEFINE('initApp()')                : (initApp_END)
initApp

**** constants
* constants identifying fields in the appData table
DIRTY_FIELD      = 10
FNAME_FIELD      = 11
BNAME_FIELD      = 12
MOVIE_FIELD      = 13
CNTRL_FIELD      = 14
CTYPE_FIELD      = 15
OPRTN_FIELD      = 16

NEWNO_FIELD      = 20
WINNO_FIELD      = 21

* values for the CTYPE_FIELD field
CNTRL_LINEAR     = 0
CNTRL_VR         = 1

* some strings
NEW_MOVIE_NAME   = "Untitled"
APP_NAME         = "SnoEez"
WIN_NAME         = "winRTM"

**** global variables
* create the table we use to store global data
appData = TABLE()

* initialize global variables
appData[NEWNO_FIELD] = 1
appData[WINNO_FIELD] = 1

* set up the screens
setUpScreens()                : (RETURN)
initApp_END
```

When we open a movie file, we execute this code to create a unique identifier for the movie:

```
winName = WIN_NAME appData[WINNO_FIELD]
appData[WINNO_FIELD] = appData[WINNO_FIELD] + 1
```

Then we store the relevant data into the `appData` array:

```
appData[winName ".", MOVIE_FIELD] = movie
appData[winName ".", FNAME_FIELD] = fileName
appData[winName ".", BNAME_FIELD] = basename(fileName)
appData[winName ".", CNTRL_FIELD] = mc
appData[winName ".", DIRTY_FIELD] = 0
```

### Controlling Movie Playback

Happily, the iPod's buttons, originally intended to control music playback, can be used to control movie playback while `SnoEez` is the active application. We can very easily bind the operation of the buttons and the scroll wheel to standard movie controller operations. For instance, to make a press on the Fast



Forward button position the active movie at its end, we can execute this code:

```
topMovie = topMovieWindow()
BIND_BUTTON(kFastForward, GoToEndOfMovie(topMovie))
```

Here, `topMovieWindow` is a user-defined function that returns the topmost movie. The implementation of this function is left as an exercise for the reader.

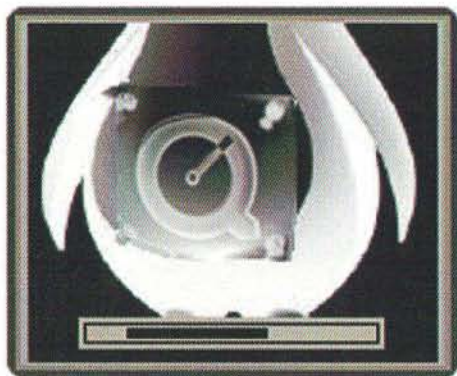
## Editing Movies

Let's look briefly at editing movies. **Listing 5** shows the `doEdit` function for handling the four basic edit operations.

### Listing 5: Handling an edit operation

```
doEdit
    DEFINE('doEdit(op)')                :(doEdit_END)
doEdit
    winName = topMovieWindow()
    mc = [winName " " CNTRL_FIELD]
    IDENT(op, "undo") MCUndo(mc)         :S(CH)
    IDENT(op, "cut") mov = MCCut(mc)      :S(CO)
    IDENT(op, "copy") mov = MCCopy(mc)    :S(CO)
    IDENT(op, "paste") MCPaste(mc)        :S(CH) :F(RETURN)
CO    PutMovieOnScrap(mov, 0)
CH    IDENT(op, "copy")                  :S(RETURN)
      setWindowDirty(winName, 1)         :(RETURN)
doEdit_END
```

The only difficult task involved in movie editing is setting a selection. The best solution I've been able to devise is to use the scroll wheel to select from the current movie time either forward or backward. **Figure 7** shows a *selection bar* superimposed on the movie screen after the user selects the Select item.



**Figure 7:** A movie with the selection bar

The code required to display and manage this bar is fairly lengthy, and I've omitted it in the interests of saving space.

## INSTALLATION

Once we've got our SNOJOB script completed, it's easy to install it and any QuickTime movies onto the iPod. The movies of course need to be copied into the `Movies` folder in the `iPod_Control` folder. The script needs to be encoded by adding 13 to the ASCII value of each character in the file. (This isn't strictly

a rot-13 encoding, since classic rot-13 alters only alphabetic characters and leaves other symbols untouched.) Finally, move the encoded script into the `Extras` folder in the iPod. SnoEez now appears in the list of iPod extras, as shown in **Figure 8**.



**Figure 8:** The Extras screen

## CONCLUSION

Supporting playback and editing of QuickTime movies on the iPod is surprisingly straightforward, once we've untangled the knots and ignored the red herrings that Apple has thrown in our path. All it takes is a little sleuthing, a little knowledge of SNOBOL, and a little persistence. No fooling!

# HelpLogic

The Help Authoring Solution for Mac Developers

**Trial Download Now Available**

Easily create help systems for your software applications and web sites from a single source.

Save time with the integrated Code Editor, Visual TOC Builder, Workshop and Link Manager to quickly generate Web-based Help, Apple Help, PDF, UniHelp and more.

**electric butterfly** [www.ebutterfly.com](http://www.ebutterfly.com)



By John C. Welch

# A Look At Panther Server

## *Digging past the hype into Apple's latest server OS*

### MAC OS X SERVER: A MAJOR REACH FOR APPLE

Now that we've all had a few months to weigh in on Mac OS X Server 10.3, I figured it was time to see how it stacks up against the previous version of Mac OS X Server and other server OS's, primarily Windows 2000/2003 Server. This is less a review of Mac OS X Server as a product, and more a look at how it does the job as a server. (Keep in mind print lead times, this article is based on Mac OS X Server 10.3.3)

### NEW FEATURES

There is a plethora of new features in Mac OS X Server 10.3, that cover almost every feature of the product at almost every level.

### Single Signon/Kerberos

For network administrators, Mac OS X Server 10.3 represents Apple's first serious attempt at one of the holy grails of any network, Single Signon. The idea for single signon is simple: you sign onto the network one time, and after that, you are authenticated for any network resource you may need. So, in a perfect single signon environment, you will log on to your Mac running Mac OS X 10.3.x (You pretty much have to be running Panther for single signon, aka SSO to work). The user id and password you provide gets you all the credentials you need for everything from network file shares to email to printing. Does Apple achieve this? Well, as we shall see, mostly.

Apple is using MIT's Kerberos security system as the basis for SSO in Panther. Kerberos is an open standard for network security, and it is designed to allow for a highly secure environment that is secure without needing a firewall. (Considering that Universities need to be highly open environments, and often use protocols that do not play well with firewalls, the need for security without a firewall is critical.) Kerberos is a network authentication mechanism. Its purpose is to validate that the person trying to get to a resource

is a known person. Other services then use this authentication to authorize that person to use that service or services. This is an important distinction. Kerberos does not allow you to get at share "foo" on AFP Server "Bar". All Kerberos does is say "whomever is authenticating as principle (Kerbspeak for user id) TAFKAP has indeed successfully authenticated (or not) as that principle." It is then up to the service to decide if that person is authorized to make use of its services or not. Kerberos is a cross – platform, standardized way to do authentication. (I'm not going to go into an explanation here, because there is an excellent functional and descriptive explanation of Kerberos in Mac OS X written by Joel Rennich at AFP548.com: <http://www.afp548.com/Articles/Panther/kerberos1.html>)

So, thanks to the way Kerberos works, it is highly compatible with SSO systems, so Apple has implemented a (mostly) stock MIT KDC in Mac OS X Server, and Mac OS X Server is now able to act as a Kerberos Domain Controller, or KDC.

### Open Directory

Open Directory, now Open Directory 2 has received major attention from Apple as well. It is now based on OpenLDAP, and uses that, along with the BerkeleyDB as a data store to provide fast, scalable, secure directory services for Mac OS X Server networks. NetInfo is still there, but by default, it is used only for local machine records, and is now considered a legacy service. LDAPv3 support is much improved, along with BSD Flat Files and NIS (Not NIS+) support. The big news here however is the addition of an Active Directory plugin, which allows Mac OS X Server to directly and more cleanly access Active Directory networks without the schema change workarounds that Jaguar server required.

### Admin Tools

The Server Admin tools have undergone yet another change. Now, actual server administration is done from the aptly named "Server Admin" application. It controls all non – user/non – client management. Along with greatly improved interfaces for the Firewall, NetBoot, Open Directory, etc. is the ability to manage multiple servers from a single window, and apply standard

John Welch <jwelch@provar.com> is an IT Staff Member for Kansas City Life Insurance, a Technical Strategist for Provar, (<http://www.provar.com/>) and the Chief Know-It-All for TackyShirt, (<http://www.tackyshirt.com/>). He has over fifteen years of experience at making Macs work with other computer systems. John specializes in figuring out ways in which to make the Mac do what nobody thinks it can, showing that the Mac is a superior administrative platform, and teaching others how to use it in interesting, if sometimes frightening ways. He also does things that don't involve computerry on occasion, or at least that's the rumor.



settings to servers via drag and drop from another server. Apple has promised to make the APIs for Server Admin available, so that third parties can write their own management console interfaces. Sybase has already released a plug in for Server Admin that allows you to manage their ASE 12 Database under OS X. Hopefully Apple will encourage the Mac version of the Windows MMC snap-in market.

Workgroup Manager gained some improvements; most notably the ability to directly browse and edit LDAP information in Open Directory without needing what is still one of the most awkward tools on the market, NetInfo Manager. With NetInfo taking a notable step back in Mac OS X Server 10.3, to continue using NetInfo Manager for LDAP administration tasks would have been very silly. Server Monitor is essentially unchanged from Jaguar Server.

The second major change in Mac OS X Server 10.3 is the addition of a plethora of command line tools for running your server. Better directory services tools, user and group management, etc. Pretty much everything you can do with the GUI tools can be done via the shell and vice versa. While the command line may seem to be the antithesis of the Mac experience, any administrator knows that a good set of command line utilities can let you get a lot of tasks done far faster than the same tools as a GUI, especially if you are stuck with dialup access.

(While Apple Remote Desktop, aka ARD has been improved, and the client portion is now a standard install for Panther, since the administrator part is not a shipping part of Mac OS X Server's default configuration, I won't be talking about it here.)

### Client Administration

The Managed Client for OS X, (MCX) services have been updated for Panther. The new feature for this is the Mobile Account, which allows for managed laptops to still function as part of an Open Directory system, even if they cannot see the Open Directory network. This is a major plus for schools and other cases where you still want to manage your users, even when they are off the main network.

### Services

New in Mac OS X Server 10.3 is the JBoss Java Application server, another sign that Apple is serious about playing in the Java market. The email server has been completely replaced with Postfix and Cyrus. This finally gives Mac OS X Server email capabilities that are as first rate as the rest of the system by incorporating robust, scalable, and secure email subsystems that are well respected regardless of platform. Thanks to the Kerberos features of Cyrus and Panther, if your email client is Kerberized, then it can take advantage of SSO for email services. This is an important upgrade as the previous mail server in Mac OS X Server was simply unsuitable for heavy-duty email services.

Windows services are upgraded with the inclusion of Samba version 3.X in Mac OS X Server. This upgrade allows Mac OS X

Server to work as a Windows NT 4 Primary Domain Controller. Other changes help create better integration for users logging in from a Windows machine to Mac OS X Server.

If you haven't gotten the idea by now, there's almost nothing in Mac OS X Server that did not get a major upgrade or at least serious attention in 10.3.x. But a feature list is only half the story. The question is, did Apple do things well or not? Mac OS X Server 10.2 was a pretty mixed bag of issues, and was, in my eyes, about 50% of the way to being what it should be. So now, let's look at how well Mac OS X Server 10.3.X is doing here.

### THE GOOD

So the first thing we should look at is what Apple did right. Well, as of 10.3.3, there's a lot to like about Mac OS X Server. At the top of the list is the mail server. The previous server was an albatross around the product's neck. It was unable to keep up with the rest of Mac OS X Server, and was simply unsuited to modern email needs. The inclusion of Postfix and Cyrus finally gives Mac OS X Server an email server that can play with any commercial product out there. Because Cyrus supports Kerberos, if you use a Kerberized email client, email can fit into the SSO realm of services. Log into your machine, start up your email client, you're in. No need to perform a separate login. This is one of the handier features of Windows XP/2000 in an Active Directory domain, and it's good to see that Apple has used it in a more open fashion. The administration UI in Server Admin is sufficient to take you from no email to a pretty solid and secure email server, and you can easily go to the command line if you need more in-depth control.

The Server Administration tools have all been updated for this new version, but the most notable change is the Server Admin tool, and the scriptability of that tool as of Mac OS X Server 10.3.3. The Server administration tools outside of Workgroup Manager were a muddled collection of half-baked attempts at putting a UI on things. Server Admin, new with 10.3.X fixes many of these shortcomings. As mentioned earlier, you can now manage multiple servers *far* easier than before, and the UI is far more capable. The DNS section has been completely revamped, and is now at least usable, (although, in the end, if you are going to muck with DNS, you still want a copy of the O'Reilly book handy, and well-read. DNS is absolutely *critical* in Mac OS X Server, and a poorly set up DNS will cause you pain that has to be felt to be believed. If you are not extremely comfortable with DNS, throw money at someone who is, it will be well worth the cost). The Firewall setup is finally worth the effort, although it still needs a bit of work to be on a par with third party tools, like Brickhouse. You can set up PPTP VPNs with ease, and LT2P/IPsec VPNs almost as easily. If you want to do a "pure" IPsec VPN, you're still going to be living in the command line, an oversight that needs to be remedied at some point. The Web server/Apache setup is about the same as in Jaguar, that is, tedious and obtuse. But, the ability to drag and drop settings between servers is a major plus, as is the fact that



Server Admin is now scriptable. It's not a complete dictionary by any means, but the introduction of AppleScript into this realm is most definitely counted as "a good thing". One continual annoyance is the inability for Server Admin to deal gracefully with servers in the list that it cannot see. Modal dialogs and application lockups are the result of having an unreachable server. Server Admin really needs to emulate the behavior of Server Monitor here, and just fail quietly. If I never attempt to manage a server that I can't see, why do I need to wait for timeouts and dialogs? Just red – light the silly thing and leave me alone. But shortcomings of the implementations of the services aside, the new Server Admin tool is a huge improvement over Jaguar's version. One final improvement is that Apple has structured Server Admin so that third parties can extend it. While the APIs haven't been widely distributed yet, Sybase has created a Server Admin module for its ASE product.

Workgroup Manager has received some notable updates, primarily based on new capabilities of Panther, and the new Open Directory access is a good start on a decent LDAP manipulation tool. You can now directly manipulate information in the directory structure outside of what Workgroup Manager lets you see. You can add pre-built settings, change existing settings, or add custom settings of your own. Dealing with the Open Directory structure this way is nicer than trying to manipulate data via the command line, but Workgroup Manager still needs a proper LDAP browser, ala the Java – based LDAPBrowser tool. You can activate Server Admin from within Workgroup Manager or vice – versa. At some point, these tools really need to be integrated into one tool, mostly so that Workgroup Manager can benefit from the improvements to Server Admin.

However, even with these notable improvements, client and machine management in an OS X network is still far harder than it has to be. To really manage client machines, you almost have to buy Apple Remote Desktop, because Workgroup Manager is only capable of the most limited machine management. So in essence, you have the "ARD Tax". There's no way to tell how computer lists fit into Open Directory. Are they a container? An OU? Short of using a tool like LDAPBrowser, or the command line directory tools, there's no way to tell. Want to change the IP address on a large group of computers? Hope you have some time for ARD or SneakerNet, that's the only way you're doing it. Apple really needs to look hard at Microsoft and Novell's directory management tools, and learn from them. Workgroup Manager is not up to the task of managing a large network.

As a first start, the Kerberos integration is outstanding in many ways. The basic setup is simple, and as of 10.3.3, most of the outstanding bugs with AFP and Kerberos have been fixed. You can make most of your services use Kerberos, (aka having them be "Kerberized"), and once that happens, you get the benefits of SSO, excellent security, and convenience. However, if you are going to need to do more with it than a very basic setup, you had better get very familiar with the MIT Kerberos administration documentation, [1.3/#documentation, because Apple has almost nothing beyond "read the man page". But, they at least give you a complete MIT Kerberos setup, so you \*can\* get more complex things done; it's just harder than it should be. If your needs align with what Apple anticipated them to be, then the setup is dead simple. If not, you're going to be spending a lot of time below the UI.](http://web.mit.edu/kerberos/www/krb5-</a></p></div><div data-bbox=)

Finally, the command line utilities. Apple has finally given server administrators a complete set of command line applications for running Mac OS X Server. Pretty much anything you can do through the UI, you can do through the command line, and there are a few things, such as setting mount style for Active Directory homes that you can't do via the command line. While some may shudder in horror at the idea of a command line, the fact is, server administrators need one. It allows you to integrate Mac OS X Server tools into other systems. By providing a full-featured command line interface to Mac OS X Server, Apple makes integrating Mac OS X Server into management tools on other platforms *far* easier. That is, by the way, a good thing.

### THE NOT-SO-GOOD

Under this heading we start with the Windows support. The big news is that in Panther, you have near one-click Windows NT 4 PDC setup. (I say near, because it's rarely that simple). However, if you want to make your Mac OS X Server a part of a Windows domain, then you can't use NTLMv2 or Kerberos for your login authentication. In fact, SMB under Mac OS X Server isn't Kerberized at all, so you can't have anyone needing Windows sharing participating in SSO. This puts a real crimp in SSO for Mac OS X. Along with that, the Active Directory plugin for Directory Services, while a good start, is not anything close to what it needs to be for heavy use. If you have a home directory in Active Directory, it doesn't mount as your home directory under OS X, but rather as an additional mount on your desktop. If you're trying to get an Open Directory Master to talk to Active Directory, that is far harder than it needs to be.

The LDAP access in WGM is too disjointed to be useful. It's very hard for new administrators to use the directory inspector to get an idea of how all of the directory information is laid out. In Active Directory by contrast, seeing how containers and OUs are laid out is quite easy, and it makes moving resources between directory structures much easier than in Workgroup Manager.

Sharing is still not as flexible as it needs to be. For example, if you, as an administrator, want to see volumes, instead of shares, Apple has a kbase article that will tell you how to do this. (<http://docs.info.apple.com/article.html?artnum=107823>). The problem is, if you follow this procedure, and you're an administrator with a network home directory, you may end up killing your ability to log in with that account, because now you can't get access to the share points. It's a binary setting, either volumes or share points, but not both. This is rather silly that an administrator can't choose how they want to get to information on the server. There's a clear need for both, and forcing you to choose like that



is silly. Since this is a system-wide change, and not a user specific change, you can't just apply it for one administrator.

The documentation, while 10.3 from 10.2 still needs a lot of work. Apple needs to include far more examples for new administrators, especially considering the lack of third party references available. For example, in the firewall docs, while they talk about setting up rules for TCP and UDP ports, they don't have an example of what a rule to allow a service into a specific machine should look like. This is one of the most common tasks an administrator does on a firewall, and it would make a lot of sense for Apple to include more screen shots of this from both the UI setup and the IPFW setup. While it's good to talk about generic hows and whys, and wherefores, for someone who hasn't been setting up Unix networks for twenty years, nothing beats a well – annotated picture. There are also far too many instances of “read the man page.” Well, the man page is only designed to tell you how a command needs to be structured. That's a syntax guide, not a howto. Since Apple isn't shipping this stuff in paper anymore, they need to spend that savings on putting more information into the docs. I really doubt they'll get a ton of emails complaining that the “documentation was just too darn informative and useful.”

### THE BAD

Now we come to my least favorite part of any analysis. The stuff that I think is just bad. Heading this list, indeed, leading it by a huge margin is the Mac OS X Server print server. First of all, it virtually ignores all the features of CUPS other than the lowest – level drivers. Adding a printer via the server only allows you to connect via AppleTalk or LPR. Um, hello, IPP? The only way to get any kind of authenticated printing is via SMB. So not only do you lose any integration with SSO, but if you want authenticated printing, none of your OS 9 clients can connect. Wait it gets better. If you have a bunch of different printers, say HP LaserJets, and you use Server Admin to connect to all of them. They all have different host names, different IPs, but the same queue name, say the default. In that case, in Server Admin, all the printers would have the same name, because queue name, not IP, or DNS name is what is used to identify the printers. Even better, if you want to set the default queue, you'll only see *one* entry. If you want to advertise the printer via Open Directory, you can't set that up in Server Admin. You have to do that manually in Workgroup Manager, by creating a manual LDAP entry. As it turns out, you can use Printer Setup Utility to add an IPP printer, but the Print Server almost totally ignores it, including to the point of not logging any of the printer activity in its logs, (which are utterly separate from CUPS. On Mac OS X Server, if you use the print server, then as far as the CUPS logs are concerned, all print jobs are created by root. Even better, CUPS *does* allow for authenticated, and even SSL printing, but the Mac OS X Server Print Server completely ignores this, as does the documentation. As of 10.3, the Print Server is simply not worth the trouble, and you're far better off with a third party product, or just getting CUPS to do it all for you.

Next on the list here is Apple's support for Mac OS X Server. It's glaringly inconsistent. Sometimes, a problem can go on for months, until you happen to talk to the right person who has the secret knowledge. It's almost like as far as Apple Support is concerned, databases are things that other people use. If you have any questions on the Active Directory plugin, or customizing install packages, that thousand – dollar support package you bought? Useless. They won't even talk to you. That is what is considered “advanced integration” and requires another support package, which *starts* at \$6000 and goes up from there, although if you have a tight budget, per – incident calls are only \$700. Apple's support organization is still *far* too weak and bush-league for the type of markets they are starting to hit. Making people keep a list of “the smart ones” is just silly. The only way I've found to get bugs into the hands of the people who will actually *do* anything about them is to get a free, online developer membership and report bugs that way.

Documentation is still far from where it needs to be. Telling someone to “read the man page” is only useful if the man page is complete. Kerberos documentation is a large gaping hole in Apple's docs. If you look at the “Command-Line Administration” documentation, the *entire* section on the command line utilities for `kdcsetup`, `sso_util`, and `kerberosautoconfig` takes up less than a quarter of page 160, and is basically a nice table telling you to read the man page. If that's all you're going to put in, why waste the space? Parts of the documentation are far from being as clear as they need to be, (tip: If you're setting up advertising a printer in Open Directory, you *have* to have a queue name. The docs make it sound optional). Accurate and complete documentation is *critical* to a server product, and Apple is dropping the ball here. The online knowledgebase is not filling the holes here at all. Searching for “Kerberos” in the Mac OS X Server section of Apple's support site results in two hits, both of which are readmes for updates.

### CONCLUSION

Even with the missteps and holes in the product, Mac OS X Server 10.3.3 is a major leap forward for the product. If Apple uses the next major version to patch and fix the problems noted here, and in other forums, they will truly have a server product that is the equal of any on the market from any company.

**MacTech®**  
M A G A Z I N E

Get MacTech delivered to your door at a price **FAR BELOW**  
the newsstand price. And, it's **RISK FREE!**

**Subscribe Today!**  
**www.mactech.com**



By Kevin Hemenway, Windusrtian Tarutaru

# Database Modification with a GUI

## Modifying our MySQL database further with GUI-based editors.

For the past three or four issues, we've been exploring the MySQL database server: how to create databases, database users and their permissions, as well as how to access data from the `mysql` command line shell, and the Perl and PHP programming languages. In our journeys, we've also seen numerous examples of how to insert, modify, select, and delete data using the Structured Query Language (SQL).

Now that we've got some initial grounding in the manual way of doing things, we can take a brief tour of how to accomplish things visually instead. The two visual tools we present in this article, CocoaMySQL and phpMyAdmin, are roughly equivalent to the included `mysql` shell: you'll still need to know some SQL to get the most use out of them.

### COCOAMySQL. FREE, OPEN SOURCED, AND OS X ONLY

The first utility we'll look at is CocoaMySQL from <http://cocoamysql.sf.net/>. Its own description is succinct enough: *CocoaMySQL is an application used to manage MySQL databases (locally or over the internet). It lets you add and remove databases and tables, change fields and indexes, view and filter the content of tables, add, edit and remove rows, perform custom queries and dump tables or entire databases.* When you open CocoaMySQL, first time or not, you'll always be prompted for your database settings (Figure 1).

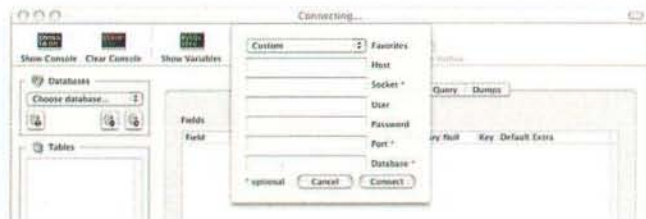


Figure 1: Configuring a database connection in CocoaMySQL.

For our **Host**, we'll use "127.0.0.1" (or "connect to the MySQL server on this machine"), and you can leave **Socket** blank. **User** and **Password** could be one of two things: the root user created when we first installed MySQL, or the specific username and password of the database we've been fiddling with. If you choose to access your MySQL server as the **root** user, you can leave the **Database** field blank, as you'll be able to choose from a master list of databases (via the dropdown menu on the left side of Figure 1). You can also ignore the **Database** entry entirely: a lesser-privileged MySQL user would be given only a list of databases they have permission to (for example, leaving **User** and **Password** blank would give you access to the MySQL `test` database created during installation). If you'd like to connect to a specifically named database, have a blast doing so. You can save your settings by choosing the "Favorites" dropdown and then "Save to favorites..."

Figure 2 shows us connected as the `davemarksman` user to the `mactech` database. The list of database tables is shown on the left hand side, and the currently selected `person` table is described in the right hand pane. Depending on the level of access your MySQL user has, you'd be able to add, modify, or delete rows, as well as indexes, for this particular table.

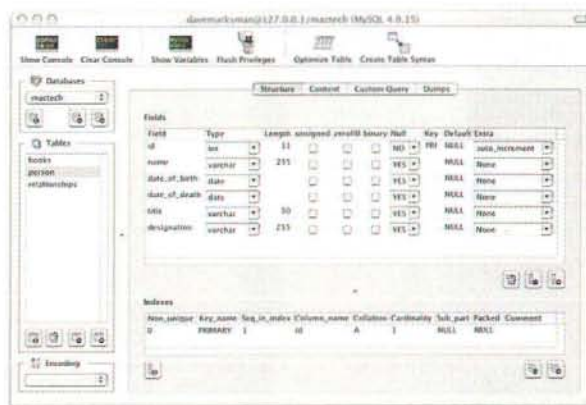
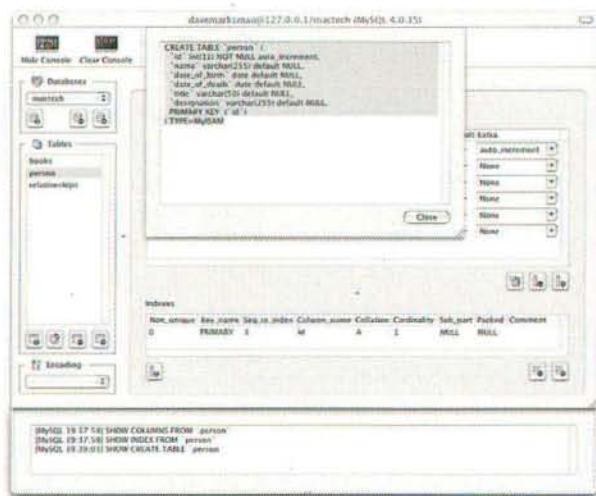


Figure 2: The person table of our mactech database.

**Kevin Hemenway**, coauthor of *Mac OS X Hacks* and *Spidering Hacks*, is better known as Morbus Iff, the creator of `disobey.com`, which bills itself as "content for the discontented." Publisher and developer of more home cooking than you could ever imagine (like the popular open-sourced aggregator `AmphetaDesk`, the best-kept gaming secret `Gamegrene.com`, the ever ignorable `Nonsense Network`, etc.), he's currently a more-than-eighth level RDM on the Phoenix server. Contact him at `morbus@disobey.com`.



Particularly handy are two of our toolbar buttons. "Optimize Table" will do as it suggests: some housekeeping to keep tables that have a regular (and healthy) flow of row updates. The MySQL documentation suggests you'd only need to do this once a week or month for the heaviest of database tables, but if you're in CocoaMySQL on a regular basis, it's hard not to flick a click in its general direction. Our neighbor, "Create Table Syntax", simply spits out the raw SQL used to create the currently selected table (**Figure 3**). You may not find yourself using this button a lot, but it's far quicker than issuing the SQL manually.

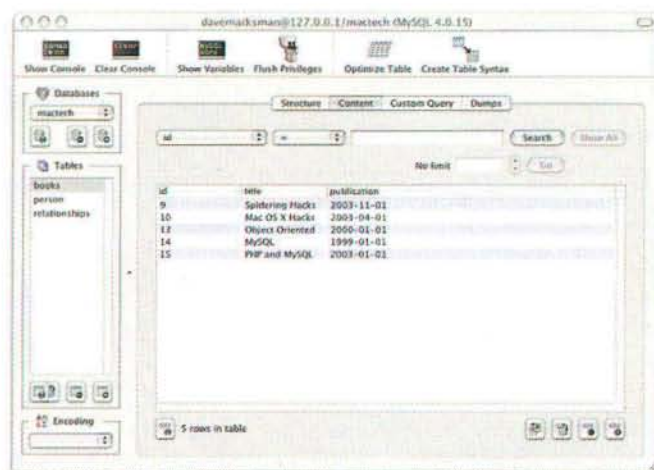


**Figure 3:** The results of a "Create Table Syntax" click.

You'll also notice another handy feature of CocoaMySQL: the console, enabled with the "Show Console" toolbar button (see **Figure 3**). It keeps a running log of every SQL statement you've issued during your mousing with the GUI. This becomes very useful and instructional when you're still a SQL neophyte.

Let's choose the "books" table from our "Tables" menu,

and click on the "Content" tab of our right-hand pane. You'll see a list of all the data within that table (**Figure 4**). You can rearrange the width of the columns as you see fit, but be forewarned that these visual-only preferences will be lost when you quit CocoaMySQL. You also have the ability to more narrowly determine what content you'd like to see with the forms above the data field: choose the table field to search through, the type of evaluation (context-sensitive, it'll change depending on whether the field is an integer or string), and the desired match.

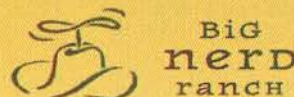


**Figure 4:** Displaying the contents of our book table.

Unfortunately, you can't create AND/OR queries here; instead, you'd use the "Custom Query" tab, as shown in **Figure 5**. Previously entered custom queries are selectable from a dropdown, can be saved as "Favorites", and SQL errors are displayed on screen. It'd be nice if a future version of CocoaMySQL could allow multiple conditions under the

## Now serving Cocoa<sup>®</sup> just the way you want it.

Training for Mac OS X doesn't have to be the same old flavor. Reserve your seat in a class at our scenic lodge location, or have experts come to you for **Extreme Mentoring**. Two weeks of on-site instruction and collaboration, customized to the requirements of your project. Book now for 2003. See why we're different.



Intensive Classes for Programmers  
[www.bignerdranch.com](http://www.bignerdranch.com)



"Content" tab, but for those that know SQL already, we'll get by with the "Custom Query" tab instead.

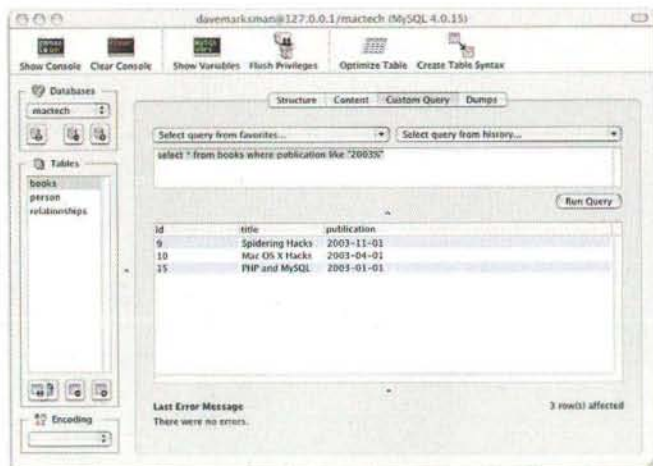


Figure 5: Custom queries within CocoaMySQL.

CocoaMySQL can also export entire databases in one of three formats: raw SQL (for when you want to dump the entire database, and then import it elsewhere), and comma-separated or XML for entire tables, groups of tables, or a custom query. The XML export, buzzwordy enough to satisfy feature gluts, is a wrapper for the same feature through the mysql shell:

```
~ > mysql -X -e "select * from books" mactech
<?xml version="1.0"?>
<resultset statement="select * from books">
  <row>
    <id>9</id>
    <title>Spidering Hacks</title>
    <publication>2003-11-01</publication>
  </row>
  <row>
    <id>10</id>
    <title>Mac OS X Hacks</title>
    <publication>2003-04-01</publication>
  </row>
  <row>
    <id>13</id>
    <title>Object Oriented Perl</title>
    <publication>2000-00-00</publication>
  </row>
  <row>
    <id>14</id>
    <title>MySQL</title>
    <publication>1999-01-01</publication>
  </row>
  <row>
    <id>15</id>
    <title>PHP and MySQL Web Development</title>
    <publication>2003-00-00</publication>
  </row>
</resultset>
```

## PHPMYADMIN: FREE, OPEN SOURCED, AND PLATFORM AGNOSTIC

If there's one major drawback of CocoaMySQL, it's that you have to be using Mac OS X. When you desperately need a quick and dirty piece of data, you'll probably be using something downright distasteful. phpMyAdmin (<http://phpmyadmin.sf.net/>) is a web-based solution that can be accessed from everywhere you want to be, and on whatever OS you happened to be saddled with. It's also one of those applications that can cause some quizzical misgivings until you become familiar with it.

To install phpMyAdmin, download and extract the latest version (2.5.6 as of this writing) into `/Library/WebServer/Documents/phpMyAdmin/`. Open `config.inc.php` in your favorite editor and modify the `$cfg['Servers'][$i]['password']` to contain your MySQL root password. Also take a look at the surrounding configuration values and set them as appropriate: the host and `PmaAbsoluteUri` are especially relevant (for our purposes here, simply inserting the root password will work well enough). Finally, load `http://127.0.0.1/phpMyAdmin` in your browser to display Figure 6.



Figure 6: The start page of our phpMyAdmin installation.

phpMyAdmin provides an insane amount of features in regards to database manipulation and analysis: you'll do yourself some good to familiarize yourself with the available documentation. For now, choose the `mactech` database from the left-handed dropdown menu, and you'll be shown something similar to Figure 7.

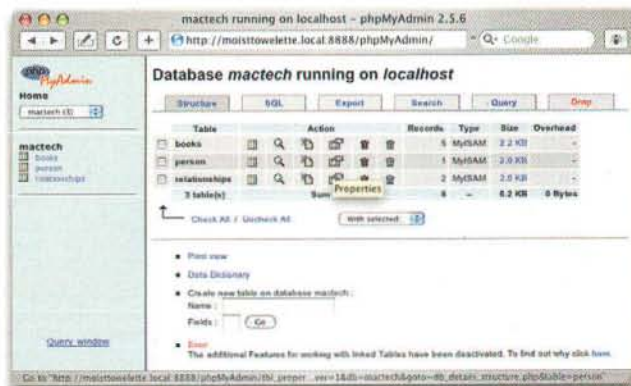


Figure 7: Icons, options, and inputs aplenty



As previously alluded, it is very easy to be overwhelmed by the sheer enormity of options available. Naturally, things will start falling into place if you use phpMyAdmin often enough, but until then, all the icons have tooltips, all the errors have clickable explanations, and most of the worrisome options have "are you sure?" Javascript confirmations. **Figure 8**, accessible by clicking the "Properties" icon of a table, is similar to the table display of CocoaMySQL's **Figure 2**, and **Figure 9** (the "Browse" tab) is equivalent to the listing of CocoaMySQL's **Figure 4**.

Database mactech - Table person running on localhost

Field	Type	Attributes	Null	Default	Extra	Actions
id	INT(4) UNSIGNED	PK, AI		auto_increment		
name	VARCHAR(255)		Yes	NULL		
email	VARCHAR(255)		Yes	NULL		
date_of_birth	DATE		Yes	NULL		
date_of_death	DATE		Yes	NULL		
sex	ENUM('M', 'F')		Yes	NULL		
designation	ENUM('CEO', 'VP', 'SVP', 'MGR', 'EMP')		Yes	NULL		

**Figure 8:** The person table of our mactech database.

Database mactech - Table books running on localhost

id	title	author	isbn	pubdate
1	The Hobbit	J.R.R. Tolkien	019-246167-1	1937-09-01
2	The Lord of the Rings	J.R.R. Tolkien	019-246167-2	1954-09-01
3	The Silmarillion	J.R.R. Tolkien	019-246167-3	1977-09-01
4	The Hobbit and The Lord of the Rings	J.R.R. Tolkien	019-246167-4	1966-09-01
5	The Hobbit and The Lord of the Rings	J.R.R. Tolkien	019-246167-5	1966-09-01

**Figure 9:** Displaying the contents of our book table.

#### HOMEWORK MALIGNMENTS

And thus completes our dissertation on basic database access with MySQL, the shell, PHP, Perl, CocoaMySQL, phpMyAdmin, and OS X. Helpful? Useless? Lacking? Confusing? Want more? Have questions? Next month, we'll be starting a new breed of *Untangling the Web* articles, but until then...

Email your suggestions, thoughts and comments to [editor-in-chief@mactech.com](mailto:editor-in-chief@mactech.com).

# New PrimeBase 4.2 Replication Server

## Check out the fully programmable Replication Server

- Bidirectional Updates supported
- Update 3rd-party DBMS
- Send Emails
- Post/get Data to/from Websites

**SQL-Runtime Plugin  
for REALbasic  
\$ 499,-**

## All PrimeBase Server Software

- SQL-Runtime Libraries available
- SQL Database Server
- Application Server
- Replication Server
- Open Server

## Available on the most popular platforms

- Completely cross-platform
- Full-text searching and indexing
- Mac OS & OS X
- Linux
- Solaris
- IBM AIX
- all Windows platforms

 **PRIMEBASE**

SNAP Innovation GmbH  
Altonaer Poststraße 9a  
D-22767 Hamburg / Germany  
[www.primebase.com](http://www.primebase.com)  
e-mail: [info@primebase.com](mailto:info@primebase.com)  
Fon: ++49 (40) 389 044-0  
Fax: ++49 (40) 389 044-44



By Guyren G Howe

# Notes from the REALWorld Conference

## *REAL Software's first conference revealed exciting new directions*

I just finished attending REAL World, REAL Software's first annual conference.

It was satisfying to meet with REALbasic developers from all over the world, and the official statements and product previews at the conference gave me confidence in the future of one of my favorite development platforms. Herewith, a grab-bag of observations from the conference.

### **VERSION 6 EARLY PREVIEW**

There were two particularly exciting revelations at REAL World. The first was an early look at Version 6.

Wow. What an improvement. An integrated IDE is among the greatest of user interface design challenges, because a computer program is a very complex structure, and the IDE needs to support the programmer in manipulating this structure in many different ways.

On top of this, REALbasic faces the challenge of working on different platforms with different user interface conventions. Worse, one of these is Windows, where the interface conventions are just broken.

While what we were shown is an early draft, it was quite clear what direction REAL is taking us. The new interface is a sort-of cross between Safari and iTunes. The default is an aggressively tabbed interface, starting from a Home tab akin to the current Project window. Everything you open comes up in a new tab by default, although you can opt for a new window, if you wish. REAL said they would like to give us options that would in sum turn this interface back into what we have now, if you didn't like the new style, but I thought it was great.

There were niceties everywhere, even in this early preview. The Project tab could be switched to an Icon view that doubled as a class browser (hooray!). The Window Editor provided a pasteboard area (akin to PageMaker), to which things could be shuffled while working on a window, and where non-control items could be placed, so they no longer clutter up the actual

window. The code editor provided little bracket lines just outside the text, to show you where each block begins and ends, and you can click on the line to select the block. There were keyboard shortcuts to enter the End If when you typed an If. And on and on and on. It was amazing how complete even this preview seemed.

The new interface, in being much more like OS X or web browsers than the OS 9 Finder style we've seen so far is much more portable across platforms, because a browser-type interface has become universal.

It was further revealed that the IDE for Version 6 was being developed with REALbasic. This is wonderful news. This means that REAL software are testing their own product pretty seriously in writing it. It makes the programmers more productive, which is why I think we're seeing such rich capabilities in the new IDE. And it gives the programmers an extra incentive to add the kind of language features that will make such a product easier to write (robust introspection and a better object model spring to mind).

There were no substantial revelations about language features for Version 6, but since REAL Software has a full-time compiler guy, who seems very smart, and very interested in improving the language, and has essentially finished implementing all the features of the old compiler, we might see some interesting new features soon (the guy has to spend his time doing *something*). I certainly heard several developers talking with Mars (great name for a compiler wizard, don't you think?) about their desire for various language features.

A side note: if you thought the new *Extends* keyword in method declaration is an interesting feature, it turns out that it is syntactic sugar for a feature that has long been available in the new compiler: you could invoke any method as a message to its first argument, so:

A regular method call

```
MyMethod(x, y, z)
```

could until now just as easily be written

The same method call, as a message to its first argument

```
x.MyMethod(y, z)
```

**Guyren G Howe** works in artificial intelligence research, after years of work as a technical writer and developer. He is married with one child, is an Australian, and lives in Austin, Texas. Guyren has been working with REALbasic for several years. Most notably, he wrote the REALbasic Curriculum Project, an extensive computer science curriculum, for REAL Software (available from the REALbasic website).



Who knew? In any event, you are now required to use the new *Extends* syntax before you can do this.

Unfortunately, a little experimentation reveals that *Extends* doesn't work quite as naturally as one might like. I was going to write a nice little debugging library by declaring the likes of a *Log* method that *Extends Object*. Then every class in the program would support a *Log* method call. Sadly, declaring a method *Sub Log(x As Object, message As String)* does not currently let you call *Log("My cool message")* from within an object; you would need to do *me.Log("My cool message")*. I mentioned this to Mars, and I hope he decides to add this little bit of rounding out for an interesting little feature.

### **SWORDFISH**

The second exciting revelation at REAL World was a new product for writing CGIs in REALbasic. The product is known for now by its code name: *Swordfish*.

What little we were shown seems great: you drag web pages into the application, and the IDE provides the ability to script the controls in the pages using conventional REALbasic code. While this may introduce minor changes into the HTML of the page, almost all of the action is server-side. Done right, this strikes me as a very savvy play: the product should produce very fast, compiled server-side code. Existing REALbasic programmers are now web programmers with minimal extra learning, and folks wanting to write server-side code who don't know a suitable language will find REALbasic more approachable than the likes of Perl or PHP. I'm not sure how big this market actually is, but I'm sure it's big enough to support a product like this pretty well.

Oh, and before you ask, it will work with Apache or as a stand-alone web server. And REAL Software is "looking into" IIS.

### **VISION**

The keynote briefly mentioned REAL Software's vision for the future. While it was mostly market speak, the part that was interesting to me was that they clearly intended to target handhelds and, interestingly, plugins (for Photoshop and the like). *Swordfish* was the first evidence, they said, of their desire to let us take our REALbasic skills "everywhere".

### **DREW CAREY**

We were shown a clip of a mention of REALbasic on the Drew Carey show. This was just too freaky for words, and elicited quite a cheer from the crowd.

### **THE DEVELOPERS, AND A PLUG**

I'd say there were 50 or so developers at the conference, quite a few from outside the US. The atmosphere was optimistic, and the diversity in the ways REALbasic is being applied by these people was fascinating. From in-house development for a medical research organization, to vertical applications, to games, simulators, educational software, utilities... It was all there. Encouraging that REALbasic has managed to be such a good tool for so many uses; I just hope the reach of the company doesn't exceed its grasp as REALbasic's ambition increases.

I found it interesting that many of the developers there clearly didn't have a computer science background, and that even some of the most experienced REALbasic developers didn't understand all of the generally useful important features in REALbasic (a number of developers there didn't understand or uses Interfaces, for example). This is not entirely surprising, as REALbasic has always been designed to be friendly to beginner programmers. Very often, it is the experts in a field who can write the best software, not professional programmers.

I will say this: if you are using REALbasic without such a background, REAL Software offers the Curriculum Project, a 30-lesson Computer Science course using REALbasic, which I wrote (search the REAL Software site for "Curriculum"). I am confident that it is an excellent way to round-out your understanding of how to use the REALbasic language to maximum effect.

### **THE SESSIONS**

The conference went for two days, with three seminars on each day. I'd like to have attended more seminars than I was able, and I heard good things about most of them. For my part, I came away with a new appreciation for the ease with which REALbasic can be used to manipulate QuickTime content, or to do REAL-time 3D graphics. The QuickTime sessions showed a simple application that nevertheless managed to replicate quite a bit of the functionality of iMovie. Something I didn't realize is that it is not necessary to have QuickTime Pro installed to use Quicktime to compress and edit movies — it seems that all QuickTime Pro does is activate these extra features in Apple's Quicktime player. For my part, I now intend to develop a program to make it easier to work with the files I get from my ReplayTV.

### **THE DEVELOPER PROGRAM**

I heard a number of positive comments about the developer program. Everyone who commented said that they'd received useful referrals from the program. REAL Software mentioned at one point that they had difficulty at times finding developers for the referrals that they had available.

### **REALBASIC DEVELOPER MAGAZINE**

It seemed that almost everyone at the conference subscribed to REALbasic Developer, and most of the presenters had written for it. I was given a free copy of the current issue, and it looks like an excellent investment in your REALbasic skills. Bi-monthly. <http://rbdeveloper.com>.

### **PRINTING: NO NEWS**

I remain disappointed that REAL has made no official announcement of features to improve printing from REALbasic. On the other hand, as I pounded on this question with developers and REAL Software folks, dbReports was mentioned repeatedly as a good solution. I spoke with the developer of dbReports at the conference, and I hope to arrange a review soon.

### **NEXT TIME: AN INTERVIEW**

Next month, I should be presenting an interview with Geoff Perlman, CEO of REAL Software.





Cool Things Happen

Conference July 12-15, 2004  
Expo July 13-15, 2004

Boston, MA

Boston Convention & Exhibition Center

Be part of the revolution at  
[www.macworldexpo.com](http://www.macworldexpo.com)



**Macworld**  
**Conference & Expo®**



# When Mac Users Meet.

## The Revolution Continues in Boston!

Connect with like-minded people and exchange your ideas, success stories and experiences at Macworld this July. Experience breakthrough products and services from leading companies devoted to you and the Mac OS platform.

We have the most innovative and knowledgeable industry experts leading our world-renowned conference programs and activities. Learn great things from the best!

**Pro Conference:** for programmers, power users, administrators, integrators and other Mac OS masters!


**Users Conference:** for educators, designers, musicians, "switchers" and new users!

**Power Tools Conferences:** in-depth training on iLife projects, FileMaker, home recording studios and other popular topics!

**MacLabs:** hands-on workshops on DVD Studio Pro, Acrobat/PDF, color management and other key topics!

**Get Connected:** Geeks & Gadgets demonstrations, MacBrainiac Challenge, Birds-of-a-Feather meetings and much more!

Discover powerful technology, make connections, and see for yourself why Macworld Conference & Expo® is the #1 event for the Mac community.



When you write code for the Mac,  
the whole community applauds your  
efforts - I mean people embrace it  
and are genuinely thankful.

Russell Miner  
Partner/Lead Developer • Small Fry Studios  
Cambridge, MA

Register online with Priority Code: **D1302**



Flagship Sponsors:

**Macworld**

**Macworld.com**



**MacCentral**



By Aidan A Reel

# Interactive Development With ActiveDeveloper

## *An Introduction to Interactive Development and a review of Inter\*ACTIVE - Technology's ActiveDeveloper*

### INTRODUCTION

This article introduces interactive development, an approach that increases your quality and productivity as a developer. It also reviews a toolset that permits this style of development to be used with Apple's development tools. Before delving into the grit of the article, a short computing backdrop explains where my interest in interactive development stems from. We will see that interactive development is not new, but has been supported by some IDEs for over 30 years. I'll outline the functionality that tools need to provide in order to be regarded as interactive development tools. I will discuss features within Objective-C that can be utilized in order to provide such tools. Finally, I will introduce, and review ActiveDeveloper (AD), a toolset that brings interactive development to ProjectBuilder and Xcode.

### BACKDROP

As a developer in the early 1990s, I coded with NeXTSTEP. NeXTSTEP allowed me to truly grasp Object Orientation. It was using instances in InterfaceBuilder that turned on the light. That phrase, 'using instances', is central to this article. I became aware that Brad Cox based his Objective-C's extensions on a language called Smalltalk. This introduced me to the language that I would use continually for the next 10 years, namely Smalltalk.

Smalltalk and its various IDEs (ParcPlace, IBM, Digitalk) taught me the power of having no compile-link-run cycle and having access to the instances within a running application. When I understood how to use tools such as workspaces, object inspectors and the stack based debugger

properly, I realized where NeXT had got its power from. Again it was working with instances. Smalltalk allows you to investigate a running program in real time. It is possible to halt execution, edit code, change attribute values on-the-fly, and resume execution. In other words, you, as a programmer can operate on your living program. You can inspect instances, traverse references, observe the actual state of your program, i.e. objects, at any point, as it runs. Having seen how the program is working (or not), it is possible to edit methods there and then in the debugger, and re-execute it. All of these activities flow from the keyboard. When you save a method, it is not only saved but compiled, and available for execution. Your workflow is not interrupted waiting for the compile and linking to complete, restarting the application and directing it back to the point of execution under investigation. You start where you left off, all in the time it takes to press **command-s**. These tools are an intrinsic part of Smalltalk IDEs.

So, after many years firmly settled in the Smalltalk industry, I had an opportunity to do some development for Mac OS X. I found myself staring at ProjectBuilder, and InterfaceBuilder. After the initial shock of how familiar they still were, and the fact that I was able to use my 1993 copy of NeXTSTEP Programming by Garfinkel and Mahoney to reintroduce myself to the developer tools and frameworks, when all was said and done I was back to the compile-link-run cycle. Regardless of the richness and maturity of Cocoa, returning to the disruption of start stop start stop programming was extremely frustrating. I soon found myself browsing the web for tools that sounded as if they supported some form of interactive development. Eventually, I found a reference to ActiveDeveloper and sat down to see what it offered.

That brings us (more or less) up to date. I am a developer whose main development experience has been supported by a language, and toolset that permitted a very

**Aidan** has finally returned home. Having wandered away from NeXTSTEP around 1995, he quietly slipped back through Cocoa's doors last year. In the meantime he worked on some of Europe's leading Smalltalk projects, gaining an appreciation for simplicity. He can be contacted at [aidanreel@mac.com](mailto:aidanreel@mac.com).



interactive style of development. Faced with an environment that did not natively support such an approach, I started to use a 3<sup>rd</sup> party product that pertained did. Now you have an opportunity to read about this style, and experiment with the same.

### WHAT INTERACTIVE DEVELOPMENT IS AND WHAT IT IS NOT

Lets start with the negative, what interactive development isn't. This is not another methodology aimed at removing identified problems in software development. Nor is it a process encouraging the adoption of a unified set of tasks that reduce the risk inherent in software development. Interactive development is a lot more humble. To the developer, it is about a new set of (easy to use) tools, rather than a new (difficult to appreciate) development strategy.

It is closer to the spirit of extreme programming in that it will enthuse the developer, and inspire them by making it easier to write robust code. As they realize that their code is improving at the same time as their productivity is increasing, their morale and confidence improves.

Before a developer can fluently use these new tools, they need to understand the mindset to be adopted. You will be pleasantly surprised to find that this mindset is very easy to adopt, and soon becomes second nature.

There are two aspects to interactive development. One is interactive investigation, and the other is interactive coding.

Interactive investigation is the task of understanding classes and frameworks by investigating instances of the classes involved. You create instances of any class, and inspect them while they are active. It is possible to send messages to their attributes and to inspect the resulting object. A tool, normally referred to as a workspace, provides a general context in which code can be written and executed. So, for example, the following code,

```
[[NSWorkspace sharedWorkspace] launchedApplications]:
```

could be written in a workspace and executed. The resulting object can be inspected in an object inspector, or browser. Inspectors provide the developer with a graphical interface to display, and browse an instance's values and references. You can traverse the various references to display other instances. Inspectors also provide a coding area whose context is the currently selected attribute. Using this context permits the sending of messages to the selected attribute. This kind of exploration and investigation cements the developer's understanding of the classes under scrutiny.

This kind of investigation normally melts into incremental development. A developer will start by reading some documentation or guessing that a particular class may suit their needs. Taking these classes, and creating instances and investigating their behavior, the developer can confirm that these classes do indeed support their requirements.

Typically, if they have written code that confirms this, then this very same code will probably be quite close to what they require. So, the next step is to move this code to a method. Note that I say move rather than convert. Because you are performing your investigation in true Objective-C, this task is typically a matter of copy & paste, changing some temporary variables into instance attributes, which is not a major task. There is no converting, rewriting, or debugging of a new piece of code.

Developers can also be faced with having to quickly understand complex classes in order to enhance or debug. With interactive investigation, the developer can work within a workspace (away from the complexity of the existing application) on salient classes, inspecting instances, traversing references, and sending messages to attributes. In effect, building up a mental picture of the instance graph. In the same way that a picture is worth a thousand words, being able to interrogate live instances is worth a thousand pages of documentation. A slight exaggeration but I think you will understand the sentiment.

Next, interactive coding avoids the compile-link-rerun cycle. In doing so, the programmer is freed to continually concentrate on the problem at hand. Being able to sustain an uninterrupted train of thought increases developer productivity, and quality of work. Imagine a writer being expected to save after every edit, issuing reformat commands, linking in adjoining paragraphs, and restarting viewers before they could see the effect of their changes. As a writer, the disruption to the creative process would be unacceptable. Yet, that is the process accepted, and expected by most programmers. Interactive coding avoids the disruption, releasing you to work at your own natural pace. It needs the support of special tools, editors that avoid having to recompile the classes just edited, re-linking modules and rerunning the application. The editors should allow you to reload changes even while the application is running and allow the developer to observe the changes, in action, immediately.

### HOW IT WORKS

#### Interactive Investigation

The good news with interactive investigation is that the underlying mechanisms used by these tools (workspaces and object inspectors) do not necessarily have to be understood by developers in order to avail themselves of their functionality.

With regard to ActiveDeveloper, your project includes a library supplied with the product suite. The process to follow in order to include this library is explained in detail in the supplied documentation. It is important to note that the inclusion of this library is only required during development; the deployed application has no dependencies on ActiveDeveloper, and as such, there is no decoupling project required after code freeze. You can confidently ship the packaged application.



## Interactive Coding

Interactive coding, as supported by ActiveDeveloper, is achieved via dynamically loading categories. Since categories play a major role in ActiveDeveloper's support for interactive coding, I have included this section as a refresher on categories. If you are experienced in the use of categories and dynamic loading, feel free to skip the next two sections.

## Categories

Categories are an extension to Objective-C. The extension is used:

- as a structural aid, permitting a developer to arrange their code into distinct sets of methods.
- to extend existing classes. Developers can add methods to existing classes when they have no access to source code.

Rather than having your class contained in one large monolithic file, it is possible (using categories) to split the class into a number of smaller files. We organize the files to hold methods that have some kind of similarity, or relationship to each other. So, for example, we could group all accessor methods together in one file, all private methods in another file, methods that implement persistency in a third file, and so on. The intention is that this separation leads to the class being easier to comprehend and maintain.

It is possible to add categories to classes that you do not have the source code for. It is by using categories that companies such as OmniGroup can extend Apple's Cocoa Framework. It is important to appreciate that since we do not need access to the source code of the class in order to extend it, we do not need to recompile the entire class in order to include the new category methods. We just compile the methods in the category. This fact has some very powerful side effects, as we will see later.

Each category can reside in its own file set (interface and implementation). The following listings display templates of category interface & implementation files.

### Listing 1: Category Interface Template

```
#import "CLASSNAME.h"
                                     Category Interface Template

@interface CLASSNAME ( CATEGORYNAME )
METHOD DECLARATIONS
@end
```

### Listing 2: Category Implementation Template

```
#import "CATEGORYNAME.h"
                                     Category Implementation Template

@implementation CLASSNAME ( CATEGORYNAME )
METHOD DEFINITIONS
@end
```

The interface file imports the class's main interface file and proceeds to declare itself as a category by virtue of the

(CATEGORYNAME) after the CLASSNAME. We then provide a list of the methods that this category will contain. The implementation file imports the category interface, declares the category name in the same manner as the interface, and proceeds to provide the method definitions.

There are some restrictions in using categories, and the main one is that you cannot add new attributes to a class. In the above interface template, we only declare methods, not attributes.

Related to this is the fact that category declarations do not actually need to provide an interface file at all. We could simply provide the implementation file.

Note that if a method with the same name resides in more than one category, it is unpredictable as to which method will get executed if called. Thus the developer of an extension should try to ensure that name clashes are unlikely. One convention is to use prefixes to any method names defined in categories that extend classes you do not own.

## Dynamic Loading

The Objective-C runtime supports the loading of category files at runtime, referred to as dynamic loading. This is a very powerful feature enabling classes to be extended during an application's lifetime. All instances belonging to the category's class now have access to those newly loaded methods, even instances created before the category was loaded.

There is no reason why a category cannot be loaded numerous times during an application's lifetime.

It is these facts that allow tools to support interactive development. The special editors mentioned earlier ensure that after the edits have been saved, the category file is reloaded into the running application.

There is an important caveat to dynamic loading and its use in supporting inactive coding editors. I will return to this when you have gained more of an understanding of ActiveDeveloper.

## Summary

In this first part of the article, we described techniques that offer a more productive, interactive, exploratory style of development, and gave them the umbrella term of interactive development. We explained that these techniques can only be adopted if supported by appropriate tools, such as workspaces (supports the execution of isolated snippets of code), object inspectors (permits the browsing of instances), and advanced editors (provides the editing and loading of category files thus avoiding the compile-link cycle).

We now turn our attention to one such tool set, available to use in conjunction with Apple's Developer Tools.

## THE PRODUCT

ActiveDeveloper is a product marketed by Inter\*ACTIVE - Technology ([www.interactive-technology.com](http://www.interactive-technology.com)), that facilitates interactive development. It runs on multiple platforms (Mac OS X, Windows WOF).



A demo version is available and I suggest, if you already haven't done so, you obtain a copy, and use it as you read along.

You will notice from the web site styling, and AD's interface, that the product has its genesis firmly in NeXTSTEP. The styling may not be to everyone's taste, but I found it reassuring to know that the product has such a long lineage.

The demo version has the normal limitation found in demos of not being able to save your work (your workspace code and editor code). It has another, more annoying limitation, a time out, apparently based on the number of explorations made during a session. I found the time out to be too frequent, and pretty frustrating. I believe the creators are aware of this, and are considering increasing the limit.

As of version 2.15, AD includes a comprehensive user guide that takes you through installation, the most frequently used tools in the package, and explaining the rules of how to structure your own existing projects in order to use the AD tools.

When AD starts, you are presented with AD's main window, the console. You will tend to use this window to observe system messages, and to start your applications. One precompiled sample application is available immediately to start investigation, the ActiveAppKitExplorer.

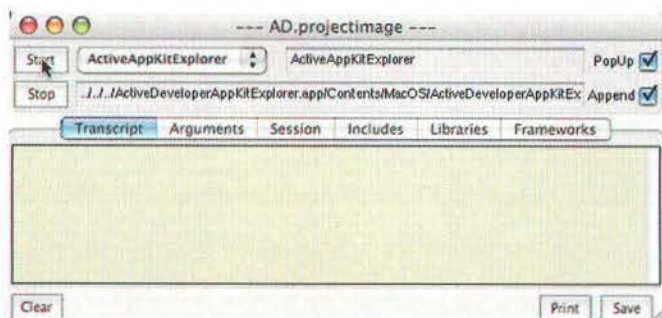


Figure 1. The ActiveDeveloper Console.

On clicking start with ActiveAppKitExplorer selected, the following window appears.



Figure 2. The ActiveAppKitExplorer main window.



# Fetch

## Mirror, mirror.

# X

## Fetchsoftworks.com

(Running dog included.)



You have just started a new AppKitExplorer application that consists of this single window, and have entered into an ActiveDeveloper session with it. I initially found the Info Panel in the window title confusing. I mistook it for the info panel of ActiveDeveloper rather than the single window of a new, separate application.

The AppKitExplorer serves as a to explore all of the Cocoa classes available to ActiveDeveloper out of the box. As you use AD, you will find that you also avail yourself of this application as a quick way to get an AD session up and running.

### Interactive Investigation

Typing **shift-command-w** from AD opens our first workspace.

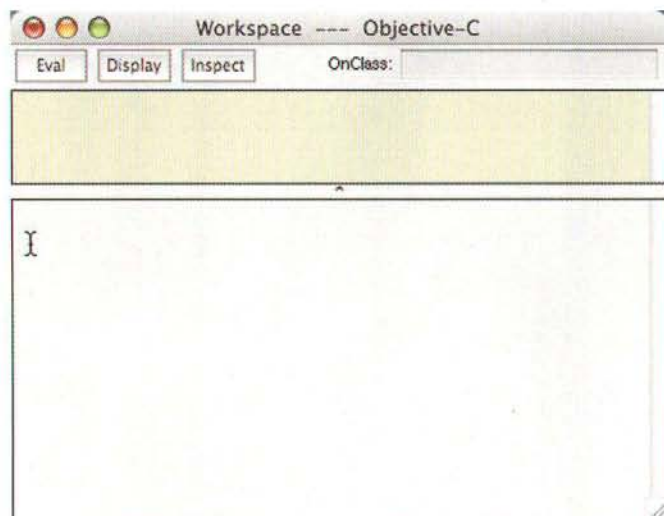


Figure 3. ActiveDeveloper Workspace.

The top pane (the import pane) allows you to add import statements, by default both `<Foundation/Foundation.h>` and `<AppKit/AppKit.h>` are imported. These imports provide the context for the lower pane (the coding pane) used to construct code. Only classes declared in interface files imported via the import pane are available for use in the coding pane. Since `Foundation.h` and `Appkit.h` are imported by default, any class from these frameworks can be investigated. The 3 buttons above the import pane are all used to execute the currently selected code in the coding pane. All execute the code, but present the results in different ways. The Eval button simply executes the code, there will be no feedback (other than that in the code) to indicate that the code has been executed. The Display button will display the textual representation of the object returned by the last statement in the highlighted code. This textual representation will appear in the Transcript tab of AD's console. Finally, the Inspect button will present the object returned by the last statement in an inspector, referred to in AD as a runtime object browser.

Remember we said that workspaces allow us to write small code samples and execute them. Returning to the example from earlier, type the following into the coding pane.

```
[[NSWorkspace sharedWorkspace] launchedApplications];
```

Note that this code has nothing to do with the application currently running. Because we have access to the Objective-C runtime via the running application, there is nothing to prevent us from creating instances from any class, as long as the class is in the context of the workspace. So the entire Cocoa API is available for such exploration. In this case, from within the workspace we have obtained an instance that refers to all the applications currently running. We can close them, hide them, etc., all from AD's workspace. This is the kind of flexibility and powerful exploration that working with instances exposes.

A word of warning from the Smalltalk community. "Do not perform brain surgery on yourself." It would be a simple matter to kill AD, or any other application, from within this workspace. I leave that as an exercise for the student! Make sure you have saved any important work beforehand.

Highlight this code and click on the Inspect button.

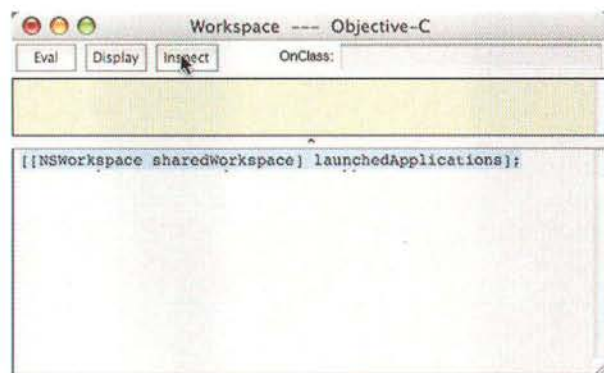


Figure 4. Executing Objective-C code in a Workspace.

A runtime object browser similar to the following will appear.

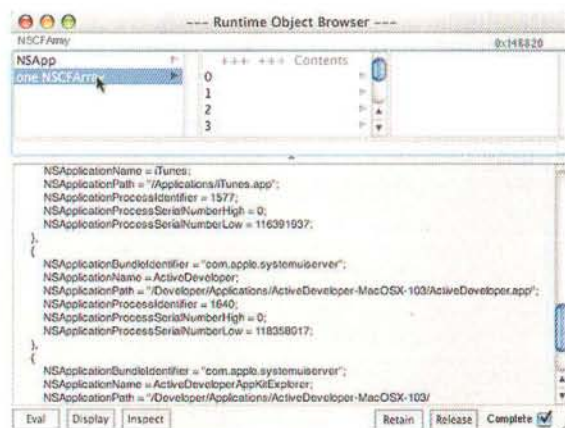


Figure 5. Exploring an instance in the Runtime Object Browser.



This window displays instances in the top left hand side browser panel, the root panel. The resulting object from the executed statement is an instance of `NSArray`, so we see such an instance displayed in our runtime object browser. The properties of the instance are visible in the adjoining column (in this case elements of the array) and by choosing these properties and traversing rightwards we can explore any instance on display. The lower pane is used to display the textual representation of the currently selected instance. This pane can also be used as a coding pane hence the appearance of the Eval Display Insect trio. The context of the pane depends on the selected instance. For example, select one `NSArray` in the browser panel, type the following code snippet after the textual representation in the coding pane:

```
[self className];
```

Now, highlight and copy this statement, then click Display. We will see `NSArray` appear in the Transcript tab. Select any item in the array, paste our copied statement after the textual representation, select, and execute our statement via Display, and we now see `NSDictionary` appear. If we repeat this by choosing an item in the Dictionary, we should see `NSString` appear in the Transcript.

What all this goes to prove is that the coding pane's context is the context of the selected attribute. To reinforce this point, click on `NSApp` in the runtime object browser, and select the `_keyWindow` attribute. In the coding pane, type in the following code:

```
[self setTitle:@"AppKitExplorer Main Window"];
```

Highlight and Eval. You should see the title of our running application change.

So, if we want to do some random exploration, we can use a workspace to create instances for us. There are more examples of this type of ad hoc exploration in the accompanying user guide.

More usually, we use the workspace to initiate exploration of the application we have just started. Going back the workspace, type:

```
[NSApp delegate];
```

Now, inspect it. The runtime object browser will display the controller in its list of objects. Again, we can traverse this object, write code in the coding pane, and continue on our Eval, Display, Inspect circuit.

It is quite easy not to realize how impressive the above examples are. In the matter of the few moments it takes to start AD, and its demo, `AppKitExplorer`, we can start to create instances of any class in Cocoa and examine it. Think for a moment of what is involved in setting up a project in order to explore a Cocoa class: setting up a new application, creating classes and methods in order to create instances, and coding a

GUI to display the results. Not to mention relying on `NSLog` statements, or setting breakpoints to use GDB to provide limited access to attributes.

### Comments On ActiveDeveloper's Support for Interactive Investigation

Overall, `ActiveDeveloper` supports the standard functions required to carry out instance based investigations. The product gives the impression that it is built by developers who focus on the technical aspects rather than on usage. This is no bad thing, but does have the downside that the product does not do itself justice in terms of it's interface, and to a lesser extent, its usability.

There are numerous user aids that could be included to enhance the developer's life. As the coding panes stand at the moment, they provide basic coding support (matching brackets). Developers have grown used to more support in terms of syntax coding, emacs like text selection, code completion, etc.. The coding panes could be improved to allow the selection of discreet areas of text, and have it Eval'd, Display'd, or Inspected from the keyboard rather than having to break off to use a pointing device. Including the commands in the popup menu would help.

When traversing the attributes in the runtime object browser, the coding panes lose their contents. It would be nice if there was some technique to retain code that was written earlier. Related to this, it would be good to have more preferences to play with. Apart from developers loving their fonts, and color schemes, they also have their own preferences as to window layout, multiple or single browsers, etc.. The preferences available could be increased.

Given the technical achievement underlying these tools, I feel a little whiny in these usability comments. Although they are important in their own right, we should not over look the fact that interactive investigation functionality is there, now, in a stable, and robust form. The usability issues do not prevent you from increasing your productivity, but you would have an even smoother investigation process if they existed.

### Interactive Coding.

AD takes advantage of being able to dynamically load a category at runtime to side step the compile-link cycle. In keeping with the rules of categories, and dynamic loading when using AD's class editor, we can load new classes, load new methods onto existing classes, and reload existing methods. Dynamic loading does not support the reloading of existing class definitions, or code that contains global symbols that are already present in the running application.

So, avoiding the compile-link cycle is achieved by placing those methods that you will be working on most in a category file, and using a special editor to do your coding.

Typing **shift-command-c** from the console brings up the editor window.



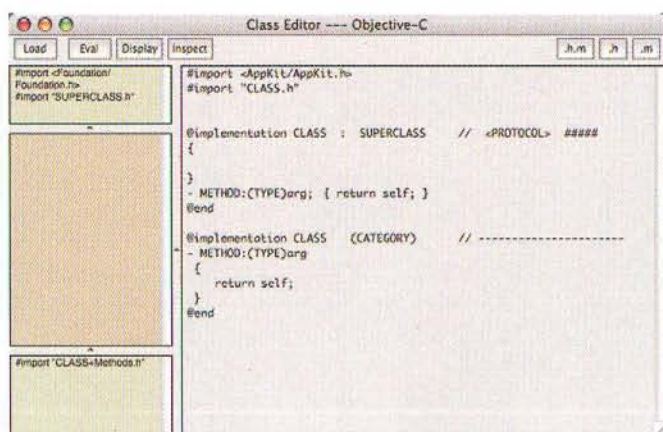


Figure 6. ActiveDeveloper's Class Editor.

We see the familiar Eval, Display, and Inspect buttons. This means you can perform interactive investigation from within the editor as you are coding. This turns out to be quite a convenient feature. There also appears a new Load button. This is used to dynamically load a category into the application that is currently running under AD's tutelage.

The top left hand pane is used to provide import statements at the start of a header file (referred to as the prefix import pane) while the bottom left hand pane is used to provide import statements that you wish to appear at the end of header files (referred to as the postfix import pane). The middle left hand pane displays the signature of methods contained in the file, and can be used to navigate to methods by clicking on the signature. Finally, the right hand pane is the coding pane.

When a class is constructed to have its interface details contained in the implementation file, AD will reward you by keeping the interface file in sync with the implementation file. This feature releases you from having to maintain both the interface and implementation files. The auto generation of the header occurs when saving the implementation file. Since saving is disabled in demo mode you will not be able to observe this feature in action.

As already stated, the documentation that comes with AD has improved substantially over the last few releases. It now contains a number of examples showing how to activate your existing projects.

### Comments On ActiveDeveloper's Support for Interactive Coding

The process of activating your own projects is covered comprehensively in AD's user guide. Once you activate two or three projects you will find that it is a pretty straightforward, almost mechanical, process.

AD's interactive coding editor utilizes the same coding pane as the object browser and workspace, providing a consistent feel between the tools. You will find working between them to be transparent.

Returning to the caveat mentioned earlier, one fact to be aware of when using categories is there is currently no way to delete a method from a running application once it has been loaded. This derives from the fact that there is no mechanism available in Objective-C to unload categories. The original purpose was to load methods into an application, and keep them there. What this means for us is that although we may have deleted a method from the category file, its last definition will still exist in the application's method space until its quits. In reality this is not a serious problem as the method will not exist the next time the application is executed. It is also possible to provide an empty method as your last edit, such as

```
methodName: { }
```

Another restriction related to the nature of categories is the fact that you cannot use AD to add or remove instance variables, or change the super class. This needs to be done within Apple's development environment. I found that this does not happen too often. Most time is spent modifying behavior rather than properties.

The same comments regarding the GUI of the class editor are applicable here. Given the quality of editors on the market, AD's coding panes are somewhat lacking. In the editor's case, this is quite unfortunate in that you will be spending most of your time in AD's editors, rather than in Apple's IDE, because of the dynamic loading support. So, you will miss functions, such as auto formatting, syntax coloring, font selection, color schemes, all the usual suspects.

### GDB and AD - Interactive Debugging

An area that I haven't mentioned yet is interactive debugging. It is possible to use GDB and AD in concert. You can attach GDB to a running application started from AD, using the application's process id. From within GDB, you can set breakpoints, etc., as normal, and have the application pause. You can use AD to edit a method, and reload it. When you use continue execution in GDB, you will find that the new edition of the method is available for execution.

It is also possible to inspect objects using the memory address displayed in GDB's value column.

This code snippet can be written, and inspected in any of AD's coding panes. Replace 0x123456 with the address of the instance you are interested in.

```
id anObject = (id) 0x123456;
return anObject;
```



# MAC OS® X PANTHER™

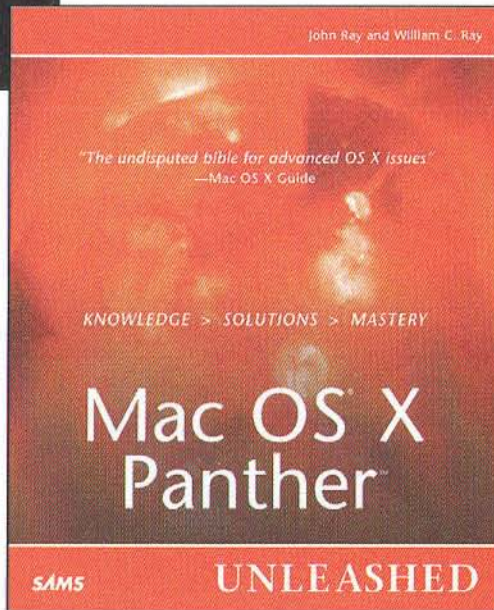
## ALL YOU NEED TO KNOW ABOUT PANTHER

***"The undisputed bible for  
advanced OS X issues"***  
**—Mac OS X Guide**

# Mac OS X Panther Unleashed

by John Ray and  
William Ray

ISBN: 0-672-32604-3  
\$49.99 US



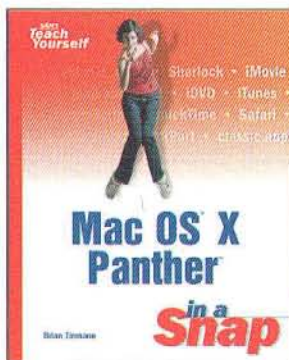
Underneath the colorful interface of Mac OS X is a powerful, complicated operating system based on BSD Unix. The third edition of Mac OS X Unleashed takes the same approach as the best selling first and second editions. Not only does this book help deal with the most trouble-prone aspects of the user interface, but also focusing to a great extent on the BSD environment and how the user and administrator can get the most out of the operating system.

**VISIT [WWW.SAMPUBLISHING.COM/MAC](http://WWW.SAMPUBLISHING.COM/MAC) FOR MORE SAMS MAC BOOKS**



## Sams Teach Yourself Mac OS X Panther All in One

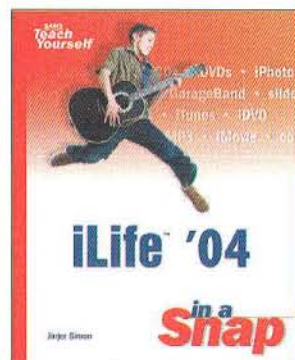
**by Robyn Ness and John Ray**  
ISBN: 0-672-32603-5 • \$29.99 US



## Mac OS X Panther in a Snap

by Brian Tiemann

ISBN: 0-672-32612-4 • \$24.99 US



## iLife '04 in a Snap

by Jinger Simon

ISBN: 0-672-32577-2 • \$24.99



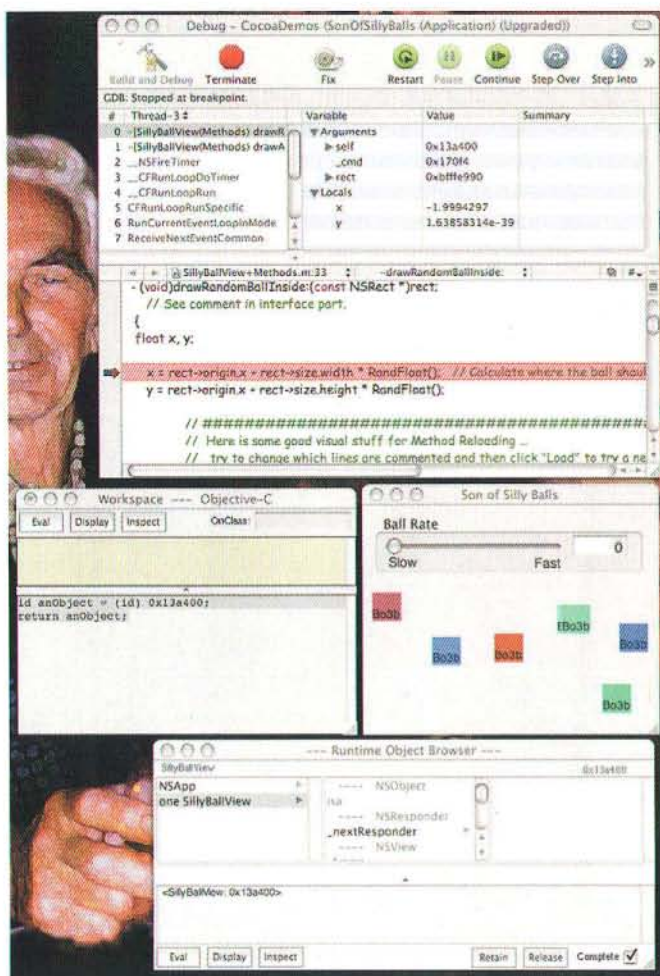
**SAMS**

[www.samspublishing.com](http://www.samspublishing.com)

amazon.com

Amazon.com is the registered trademark of Amazon.com, Inc.





**Figure 7.** GDB and AD in Concert. Note the address in GDB, the code snippet in the Workspace and the instance displayed in the object browser.

This leads to a very flexible debugging environment where you can work, as it were, in three dimensions. The first being the time line of the stack trace, the second being on-the-fly code changes, and having the debugger execute it when the application is continued, and finally, the 3<sup>rd</sup> line of attack being the ability to completely traverse live instances in object browsers.

#### CLOSING COMMENTS

I have taken you, at quite a pace, through the essence of interactive development, and the tool functionality needed to support it. Within the Objective-C arena, I explained the language features that permit such tools to exist, and introduced you to one such toolset. Finally, I now present my ranking as to how well I feel AD compares to the interactive development tool set provided by Smalltalk IDEs.

Some interactive development tools are missing, such as a true Smalltalk style debugger, but there are fundamental

differences between the Objective-C runtime, and Smalltalk runtimes which mean that certain Smalltalk tools, such as the debugger, can not be replicated completely. So this is not a criticism of AD more than a reality check. So, for features supported I would award 9 out of 10.

From my usage of the tools over the last few months, I have found it stable and reliable, and I use it constantly. Out of 10, I would give ActiveDeveloper's tools a rating of 8 just for the sheer power that the functions it supports provide. A 10 would be easy to award once the GUI supports more of the functions that developers have come to expect.

The documentation does not cover all the functionality provided by AD. There is little discussion on what the other tabs in the console are for, nor how to use the buttons, such as retain and release, in the object browser. Little mention is given to the ability to start Interface Builder from AD, and use AD to investigate the inner instances of IB as it is running. You find it mentioned in the FAQ page. But the documentation does achieve the most important goal, that of explaining how to set up your environment, and how to use the most important features. It takes you, in a logical sequence, through the most important functions, and by the end you will be up and running with your own projects activated. Marks here are 7 out of 10.

As an aside, retain and release allow you to quickly add/remove an instance to the root panel of the object browser. If you know you will need to revisit a selected instance buried deep in an object graph, by pressing retain, the selected instance will appear in the root panel. It is now at hand for future use. When it is no longer of interest, select it, press release, and it disappears. A very useful feature.

Product support has been excellent, all queries were answered promptly, and I feel there was genuine commitment to the product. Nine out of 10 for product support.

So an overall score of 8, which is well deserved.

In closing, I hope you enjoyed reading this article and found it easy to understand. More over, I hope that you have been introduced to something that will have a lasting beneficial effect on your development style.

**MacTech**  
MAGAZINE

Get MacTech delivered to your door at a price **FAR BELOW**  
the newsstand price. And, it's **RISK FREE!**

**Subscribe Today!**

**www.mactech.com**



By Ron Davis

### WEBOBJECTS FOR MAC OS X

*WebObject for Mac OS X* by Joshua Marker is part of the Visual QuickPro Guide series from Peachpit Press. This is the first book in that series I've read and if it is representative of those books I'm impressed. Just paging through it I was struck by how many pictures there were. Every page seems to have at least two screen shots. Subjects are broken down into small pieces. It is very step by step and the steps aren't two pages long. There are numerous side bars that explain related information. The layout of the book makes it easy to scan through and find information.

The first couple of chapters are introductions to the system and Project Builder. The first chapter of the book covers exactly how WebObjects works, explaining how servers can be set up and how requests are handled. If you've had anything to do with WebObjects before, or even read Apple's introduction to WebObjects, you may be able to skip the first chapter. It also gives an overview of the documentation available for WebObjects and how to access it. The second chapter covers project creation and compiling and running WebObjects apps. If you have used PB before you can probably skip the second chapter.

With WebObjects the programmer creates Components. A component is a collection of files and code that define a web page or group of pages. WebObjects uses a UI building tool called WebObjects Builder. Chapter three discusses how to use WOBuilder and how to connect code to UI. There are a lot of similarities between Interface Builder in the Cocoa world and WOBuilder in the web world. They are basically the same thing for different user interfaces. Also WOBuilder connects to Java objects, while IB connects to Cocoa objects. Used to be WOBuilder connected to Cocoa objects, but Apple unfortunately decided to change to Java over Cocoa for web applications.

A Component represents a web page or group of web pages and it is made up of *elements*. Chapter four starts explaining elements. Some elements are static, really basically straight HTML, and some are dynamic, hooking up to Java code. There are a lot of dynamic elements and this chapter is the longest in the book. It includes a section on WOConditionals, which are special elements on a component that display different elements depending on a condition.

When WebObjects was first created back in the Next days, the thing that made it so incredibly cool was EOModeler. EOModeler is a method of mapping database tables to Java objects. It used to be mapped to Cocoa objects and there were many client/server applications written with Cocoa with a database backend, but again Apple abandoned EOModeler on Cocoa.

Chapter five walks through creating a database using OpenBase, which comes with WebObjects, and creating the objects related to that database. Chapter six then connects these

objects to elements and creates a dynamic web application for changing the database.

In order to map a database to objects the relationship of different tables and rows in the database have to be mapped to the objects. Chapter seven teaches how this is accomplished using WebObjects and EOModeler.

Now that there are relationships, chapter eight talks about fetches. Doing queries on a database is how a web application gets its information. Since WebObjects is mapping objects to databases there is not a direct connection to SQL queries. Instead WebObjects provides fetches, which are how the programmer tells which database rows to create objects for. This chapter talks about how to specify fetches, how to optimize them and how to display the results. Much of this can be done visually with the EOModeler application.

All web applications are a little complicated and confusing. This is because there is no event loop in a web application. Instead each request for a page is the "event." It is the sum total of what happens from the web server's point of view. What web applications have to do is create an artificial session that keeps running even when there isn't a request coming in. In the simplest case of a web application you don't need a session, everything is done in response to one request.

Chapter nine discusses how WebObjects handles the session. In order to maintain a session each time a request is made, some piece of data has to come back with an ID for the session it is connected to. Then WebObjects goes and finds the process still running with that session ID. This information can be sent in the URL as a parameter or it can be stored in a cookie. This chapter has a side bar on this and discusses session management.

Chapter ten covers editing contexts. If a web application has a number of web forms that gather information to put in a database they may cover a number of pages. This means the data has to be kept around until all of it is collected and then committed to the database. While a session will keep the data around an editing context manages the EOModeler objects created during the session until the application is ready to commit them to the database.

If an application doesn't need session information, WO provides Direct Actions which handle everything in a one call. They are the subject of chapter eleven.

Chapter twelve talks about integrating with SSL, JavaScript and cookies. The last chapter covers deploying the application, including deploying to Linux in addition to Mac OS X.

This book is a great beginner's book for WebObjects, one that is sorely needed. It isn't really intended for people who have a lot of knowledge of WebObjects and wouldn't make a very good reference. The content and style complement each other to make it a very useable book for learning.



By Guyren G Howe

# Object-Oriented Programming with REALbasic

## *Getting the most from a unique Object-Oriented paradigm*

Welcome to the first issue of my regular *REALbasic Best Practice* column. Each month, I'll tackle the question of how one can get the most out of REALbasic. I'm going to mix things up: reviews, advice, adapting general programming techniques to REALbasic, but always with the theme of "best practice".

I'm going to start with something I touched on in last month's review: REALbasic's unique object-oriented programming feature, that I call the *Event Model*. This article assumes you are familiar with the basics of object-oriented programming.

### A LITTLE HISTORY

A little history will help us to understand where we are.

REALbasic's original designer, Andrew Barry (who sadly is no longer with REAL Software) came up with what I believe is an entirely novel, and powerful object-oriented programming model for the very first version of REALbasic, which he prosaically called *Events*. To distinguish this feature in general from particular events, I'll call the feature the *Event Model*.

Because it was too different from what folks were used to, and because it made porting code from other languages more difficult, Andrew bowed to pressure, and *also* provided support for basically the same OOP model that Java supports (single inheritance with *interfaces*).

Once Andrew left, the programmers and managers at REAL Software have, to my mind, quite reasonably focused on the many other ways in which they wanted to improve REALbasic, and they've done a bang-up job of that. But they have, in some ways unfortunately, lost focus on this cool language feature. It's still there, it still works just fine, but REAL doesn't correctly document it, they don't much talk about it, and they haven't extended it in a few obvious ways that would make it just about perfect.

### WHY YOU NEED EVENTS

You need the Events Model because it makes code development and maintenance easier, and because the result of code changes in a superclass under the Events Model is much more predictable than with traditional method overriding.

In fact, the Event Model brings such clarity, and maintainability benefits that, except for doing one particular thing, Events are *superior in every way* to traditional method overriding.

### EVENTS VS METHOD OVERRIDING

I will now explain what the Event Model is, and provide an example to demonstrate the advantages I just mentioned.

What we're about here is the relationship between the code in a superclass, and the code in a subclass. In Object-Oriented Programming languages, one of the most interesting problems is how the language should support code re-use down through a class hierarchy. We want to be able to provide some code in a superclass, and then have code in a subclass somehow extend or modify that code — and to perform further modification, and extension of behavior on down a class inheritance chain.

Every mainstream Object-Oriented language does this by allowing a subclass to *override* methods on its superclass, entirely replacing that code, and then allowing the subclass to invoke methods defined in the superclass, including quite often invoking the superclass's version of the overridden method. A common pattern will be a method on a superclass that provides code to complete some but not all of some task, with the expectation that a subclass will override the method, carry out the rest of the task, and at some point call the overridden method to do the common part.

Notice that in this scheme, the subclass is entirely in charge of the final carrying-out of the method. In particular, it is up to the subclass to determine when and how to invoke the code in the superclass as part of its design for the complete method. In an inheritance chain, control is usually passed in this way from the bottom up, with a subclass invoking the most immediate superclass's methods as it desires.

**Guyren G Howe** works in artificial intelligence research, after years of work as a technical writer and developer. He is married with one child, is an Australian, and lives in Austin, Texas. Guyren has been working with REALbasic for several years. Most notably, he wrote the *REALbasic Curriculum Project*, an extensive computer science curriculum, for REAL Software (available from the REALbasic website).



There are two serious problems with this model:

Good object-oriented design will push as much code as is reasonable toward the top of the class hierarchy, but that code has to be written without being able to rely on when and how it is being invoked by the subclasses. Even in code you're entirely writing yourself, you have to remember what conventions you intended to employ in this relationship, possibly when you come back to code six months after you wrote it; and

It is easy for a subclass to override a superclass in such a way that, although you want to make a change in behavior that is the superclass's responsibility, you can't do it without also having to change some or all of the subclasses' code as well.

The Events Model, on the other hand, turns this whole thing on its head, allowing the superclass to provide subclasses with explicit opportunities to act, and letting these opportunities pass *down* the inheritance chain rather than up it. This very neatly avoids both of the problems mentioned, as we will see.

### AN EXAMPLE

Let's consider a simple example that clearly illustrates the advantage of the Events Model.

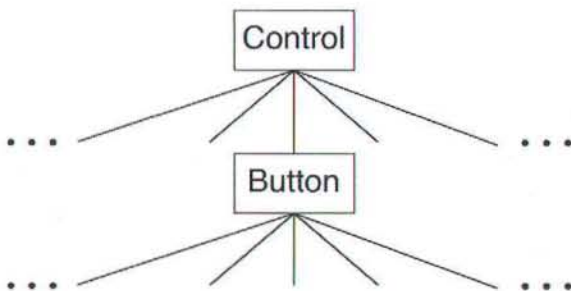
Assume we're writing an application with a non-standard user interface, so we use custom controls drawn using graphics primitives.

We create an abstract *Control* superclass, whose task it is to keep track of where the control is on the screen. We give it a *Draw* method.

Next, we have an abstract *Button* subclass of *Control*, with code to draw a common background color and some other graphic details shared by all the buttons in the program.

We then create a whole variety of button subclasses for various tasks.

So we have an inheritance hierarchy that looks like this:



**Figure 1:** A simple inheritance hierarchy

### The Method Overriding Way

When we use method overriding as our means of extension, as we go up the chain, each class overrides its superclass's *Draw* method and invokes that method as part of its own *draw* process.

This means that a control at the bottom of the chain will have a *Draw* method that looks something like this:

```
Pseudo-code for a Draw method at the bottom of the inheritance chain
Sub Draw
  Super.Draw
  ... code to draw specifics on top of background provided by
  super
End Sub
```

*Button.Draw*, in turn, has code like this:

```
Pseudo-code for Button.Draw
Sub Draw
  Super.Draw //Work out where I'm drawing
  ... code to draw background
End Sub
```

*Control.Draw* has code like this:

```
Pseudo-code for Control.Draw
Sub Draw
  ... code to work out where to draw
End Sub
```

This is all fine. We get nice abstraction and code re-use, all the usual OOP goodness.

But a problem arises when we make changes to the superclasses. Let's say we now want to make all our buttons have an *Aqua*-style highlight: we want to take whatever the subclass produces, and run a graphic filter over that to make it look like it's sitting in a drop of liquid.

We *simply can't do it* without wholesale code rewriting. With this inheritance scheme, the superclass has *no way* to obtain another chance to act after the subclass is done calling on it. The only solution is to make changes to *every* bottom-level subclass, even though we really don't need the subclasses to do their part of the drawing task any differently. A *change in the shared logic* can't be implemented in the *shared code*.

### The Event Model

An Event declaration is essentially a declaration of the interface for a method that subclasses can implement (this "method" is called an *Event Handler*), that can only be called from the declaring superclass. Once declared, the superclass can invoke the event handler just like a regular method or function.

When an event handler is called, the *most immediate subclass* implementing a handler for it gets to act. If no subclass implements a handler, the call is ignored (if the call is to a function, a "null" — 0, *Nil*, *False*, the empty string, and so on — value is returned).

Note that once a handler for an event exists in the hierarchy for a particular class, further subclasses don't even see the event. Frequently, the subclass will do whatever it needs in response to the event invocation, and then invokes



a new event it has declared, having exactly the same name and arguments, providing further subclasses with the opportunity to act.

Note that rather than the ill-defined mess that is method overriding, the language is providing a well-defined *protocol* for superclasses to delegate specific responsibilities down to their subclasses.

A good example of this scheme is the REALbasic *EditField* control, which declares a *KeyDown* event as a function with a string argument, returning a *Boolean* value. The event indicates to a subclass that the user has tried to type something into the *EditField*. If the subclass provides a handler for *KeyDown* and returns *True* in that handler, the keystroke is ignored. Otherwise, it is displayed in the *EditField* normally.

This scheme makes it easy to extend the *EditField* to create a control that, say, only lets the user enter a number.

### The Button Example, Using the Event Model

Back to our example of custom buttons.

Under the Event model, rather than each subclass overriding its superclass's version of *Draw*, only the top-level *Control* class has a *Draw* method. In addition, *Control* declares a new *DrawEvent* event. Each subclass will provide a handler for the event, and will also declare a new version of the event, calling it at the appropriate time. So now we have:

*Control.Draw* has code like this:

```

Pseudo-code for Control.Draw
Sub Draw
  ... code to work out where to draw
  DrawEvent //Ask subclass to now draw itself there
End Sub
  
```

*Button's DrawEvent* handler looks like this:

```

Pseudo-code for Button.Draw
Sub DrawEvent
  ... code to draw background
  DrawEvent //Ask subclass to draw its content atop the background
End Sub
  
```

Finally, the class at the bottom of the chain just does its part of the task.

Pseudo-code for a *Draw* method at the bottom of the inheritance chain

```

Sub DrawEvent
  ... code to draw specifics on top of background provided by
  super
End Sub
  
```

Now what happens when we want to add the new graphical tweak? We only have to change the code in one place, *Button's DrawEvent* handler, which now looks like this:

```

Pseudo-code for an extended Button.Draw
Sub DrawEvent
  ... code to draw background
  DrawEvent //Ask subclass to draw its content atop the background
  ... code to draw water on top
End Sub
  
```

Once you *get* the Event Model, you miss it in every other object-oriented language.

There are many advantages to the Event Model. For example, if you're working with a team of programmers, or you're shipping a framework for others to use, with method overriding, you need all sorts of tortuous documentation about how subclasses should interact with their superclasses. With the Event model, the language itself enforces the relationship. So not only does the Event Model make maintenance easier, but it better explicates the logic of the program.

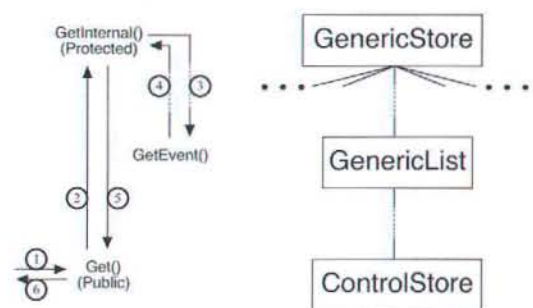
### THE EXCEPTION THAT PROVES THE RULE

...except that there is one feature for which we still need overriding: functions for which the subclass should return a more specific type than the superclass does. The canonical example of this is a generic data structure. We can write, say, a generic data store base class that stores *Objects*. But this is needlessly difficult to use as it is, because we have to cast the objects we get from it to a more specific type in order to use them, every single time we use this class.

What we want is to extend the generic data structure to one storing objects of more specific types — a list of controls, for example, where the casting to the correct return type is done for us. Unfortunately, the Events Model provides no way to do this, because the function that yields the objects must be declared to return values of type *Object* (or perhaps *Variant*).

The method overriding paradigm provides what we need here. We create a *Protected* function toward the top of the inheritance chain, then declare and invoke suitable events so that subclasses can carry out the function. Then we create a 'public face' on that function in the bottom-level class in the form of a function that returns the type we want, and have it just call up to the protected internal function, at the top of the class hierarchy, that provides the generic functionality.

So the call sequence goes like this:



**Figure 2:** Using Method Overriding to Create a More Restricted Type

We invoke the public *Get()* function on the bottom-level *ControlStore* class.

*ControlStore* invokes the protected *GenericStore.GetInternal()* function on the base *GenericStore* class.



*GenericStore* invokes its *GetEvent*.

The most immediate subclass declaring a handler is *GenericList*, which fetches the result, returning it to *GenericStore*. *GenericStore* returns the result to *ControlStore.Get()*.

*ControlStore.Get()* casts the result to a *Control* and returns it to the original caller.

Interestingly, notice that in a sense, we aren't actually extending the *GenericList* class's behavior: we're actually restricting it. All of this calling up, and back down the inheritance chain is a little weird when you first do it. Quite often, in the bottom class, you will be creating the public function, calling the protected version in the base class, then turning around and handling the event again the same bottom-level class. When I first started doing this, it felt a bit *baroque*, to be going up, and then down the inheritance chain like this, and having the public function actually fulfill its role in the event handler. But I'm quite comfortable with it now because I don't see overriding as an extension mechanism at all any more.

It might help to see that in fact, you could do this with just Events by creating a wrapper class that does the type casting. I just prefer to do it all within the same class hierarchy, because it's cleaner and clearer to do this rather than create the generic store, the wrapper class, and suitable Interface.

It would be nice if REALbasic had templates, or if some other mechanism could be found that let us stick entirely to the Events Model without any of that, but this constrained use of overriding is just fine for now.

#### IMPROVEMENTS WANTED

I'd like to see REAL Software emphasize the Events Model in their documentation. I'd also like to see them look into making some improvements to the Events Model to make it more powerful.

One obvious improvement would be to provide a compiler directive that could execute different code depending on whether an Event has a handler available or not. A common style of programming you'll want to do with Events is to provide default behavior at some point in the code for a class, but allow a subclass to extend or replace that behavior (as the *EditField* does with the *KeyDown* Event).

The only way to do that right now with Events is to call a function-style Event Handler, and do your default behavior if you get back the "null" response. But in some circumstances, you need to treat the null value as a result, rather than an indication that you didn't provide an event handler, and you wind up doing all sorts of gymnastics to work around the problem.

#### CONCLUSION

I hope I've convinced you of the virtue of the Events Model. Once you "get it", in my experience, you'll wonder how you ever lived without it.

## Long Distance

**3.9¢ Per Minute!**

**Straight 6 second billing increments**

**Excellent rates on intrastate, intralata/toll calls and international calling with no term contract.**

**Toll Free (800/888/877/866) service, same low per minute rate for incoming calls.**

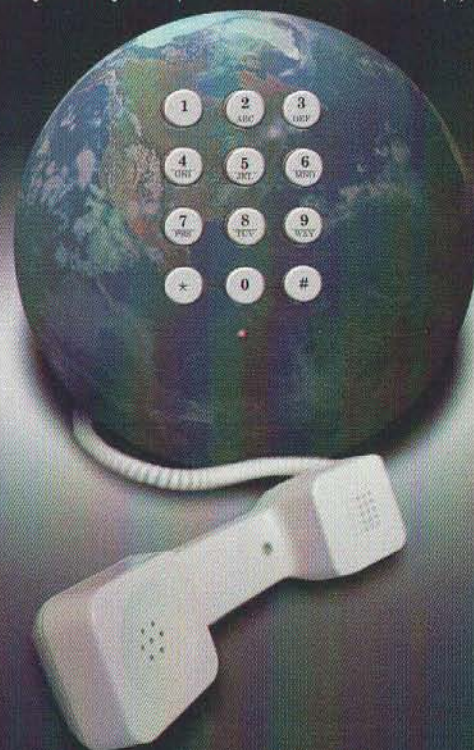
**10 cents per minute calling card.**

Detailed billing directly from Capsule Communications, a Covista Company.

**Quality electronic and telephone customer support.**

**No monthly billing fee if you sign up for AUTOPAY billing option or if your bill is over \$20.00 each month.**

(NOTE: \$1.00 billing fee is charged when your bill is under \$20.00 for all non-Autopay customers.)





By Sean Whelan

# Testtrack Pro 6.1

## *Defect tracking for distributed development teams*

### OVERVIEW

Testtrack Pro 6.1 is a defect tracking solution for software development and quality assurance teams that offers user-customizable fields, a workflow audit trail, powerful data analysis, and on-demand reporting. It is built to scale up for large teams who collaborate on projects across multiple locations and time zones. TestTrack Pro is a client/server application with client access from Web browsers and server support for OS X, Solaris, Linux, and Windows. In addition to Web browser access, there is a Windows client application. The system offers customizable workflow actions, automated E-mail notifications, import / export utilities and a SOAP SDK for simplifying integration to other applications.

Rather than just a defect tracking solution, Testtrack is aimed at filling the role of a software development life cycle Workflow Management Module. The combination of automated notifications, Source Code Control integration, and integration to other customer-facing Seapine modules (Solobug and SoloSubmit) that capture end-user reported software issues add up to a tool that can be used by all teams involved with a software release from the Subject Matter Expert and the Project Manager to the Support, Development, and Quality Assurance Teams. User security is set by Usergroup to control the granting of user rights and user access down to the individual field and command level.

We originally implemented Testtrack in the late 1990's in its earlier form, Testtrack for Workgroups. We have developers and QA staff who work both on Mac and PC machines, depending on which product line they are supporting. At the time, Testtrack was one of the few commercial defect tracking solutions that offered full clients for both Mac and Windows. With the release of Testtrack Pro in 2000, Mac users were only able to access the server

through the Web client. The earliest versions of the web client were not full featured and caused some distress for our Mac teams. Over the last four years, Seapine has enhanced the Web client to the level that it has become the preferred method of access for many of our engineers, regardless of platform. Seapine currently has no plans to re-introduce a Mac-specific client, but will continue to serve all platforms with the web client.

### INSTALLATION

OS X installation is straightforward and fairly simple. A disk image file is automatically extracted from a .gz archive and mounted on the desktop. The mounted disk image contains a typical installer and a .PDF file. The installer prompts for Admin credentials and then offers the user two paths for proceeding, an "Easy Install" and a "Custom Install." The easy install will load all Testtrack components automatically including the License Server, License Server Admin Utility, Application Server, Web Server Admin Utility, Sample Database, and the stand-alone customer issue reporting tool, SoloBug. The Custom Install allows the user the freedom to specify which components are to be installed, what server ports are to be used for communication, and other configuration details that are set up as default values in the Easy Install.

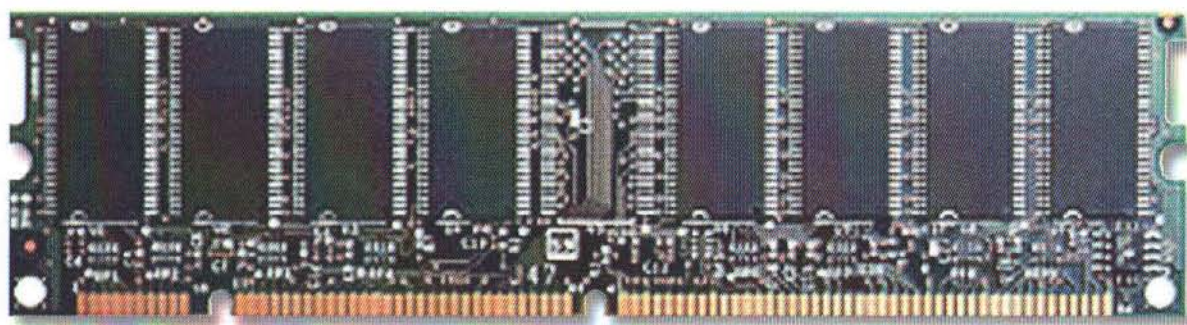
Simple configuration of the server must be done to enable the Web and License Servers. The user must go into the System Preferences and set both applications to automatically start upon machine startup. Admin User and License information will need to be keyed in to the License Server Admin Utility before logging in to the sample database or creating a database for your project.

Seapine sells its licenses per user seat and offers the choice between named licenses, and "floating" licenses which correspond to concurrent user access to the server. Prices per user start at \$295 per named user and \$795 per concurrent user.

**Sean Whelan** is the Senior Director of Quality Assurance for Commercial Print Products at EFI, Inc. He leads a QA team whose engineers test and release software targeted to customers on Mac and Windows platforms as well as testing hosted Web Products.



There are some things in life you  
can never have enough of...



Built-to-order. Lifetime Guarantee. Competitive Pricing.

**DEV  
DEPOT®**

***[www.devdepot.com/memory](http://www.devdepot.com/memory)***

Voice: 877-DEPOT-NOW (877-337-6866) • Outside US/Canada: 805/494-9797  
E-mail: [orders@devdepot.com](mailto:orders@devdepot.com)



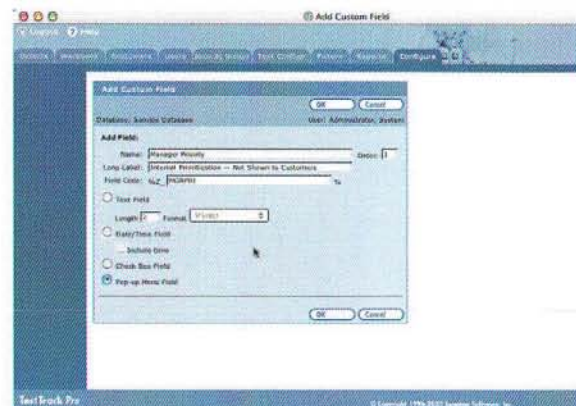


**Figure 1.** License Server Admin

## CUSTOMIZABLE WORKFLOW

One of the key features of Testtrack Pro 6.1, is that it allows Administrators to fully customize the product to fit the workflow needs of the development, support, and quality assurance groups who are using it. Administrators can customize field names, contents of pop-up field entries, field relationships, workflow stages, and the default values of fields. Customization includes specifying whether or not a given field is required for record entry, auto-assignment rules, e-mail templates, as well as record filters and reports. Testtrack allows the user to capture additional information beyond the standard defect tracking fields through the creation of unlimited custom fields. Two Custom fields can be added to the main screen, while additional fields are added to the Custom Fields tab. Custom fields are not restricted to text values, but can be added with any of the standard field types, including: date, integer, text entry, and pop-up menu with customizable entry values.

Customization is achieved by selecting options from stripped-down, easy-to-use configuration screens. The customization is so simple that you could have junior level support staff customize your database (not that you would want to, but, it is that simple.) Another advantage offered by Testtrack Pro as compared to some other defect tracking systems, is that it is not necessary to take the server offline while adding and editing custom fields or field values. This work can be done while users are logged in, which can cut down on the time spent doing Admin tasks after-hours and minimize the need for planned service outages to make updates. The only customization tasks that require a lock-out of all users are deletes of fields or record entries, as they may be in use at the time of the deletion.



**Figure 2.** Adding Custom Fields

## WORKING WITH TESTTRACK PRO

The web server is typically hosted as part of your intranet, and users in your domain can reach the service from a URL that you configure in the System Preferences of your server, such as `http://testtrack.mycompanyname.com`.

Upon Login, the user sees a relatively clean home screen that offers navigation links across the top to screens for Defects, Workbook, Customers, Users, Security Groups, Test Configurations, Filters, and Reports. Along the left side are links for actions that can be taken on whatever record or list of records are displayed in the main screen, such as Assign, Fix, Verify, and Close. The user only sees actions and links to which he/she has security rights. A developer user may be set up with the right to Fix defects, but not to Verify them, a QA engineer might have rights to Verify defects, but not to Estimate a Fix.



**Figure 3. Main Defects Screen**

Defect Entry is straightforward. The combination of industry standard fields such as Summary, Description, Severity, and Steps to Reproduce can be augmented with the addition of whatever custom fields your workflow desires. New to the 6.0 version is the ability to customize the workflow steps that a defect goes through during its life cycle. In the past, Testtrack Pro limited records to the main states of Open, Fixed, and Closed, with subsets for review status and staff





assignment. Now system administrators can enhance the workflow to include whatever additional steps might be necessary, such as Waiting For Build, or Pending Management Review.

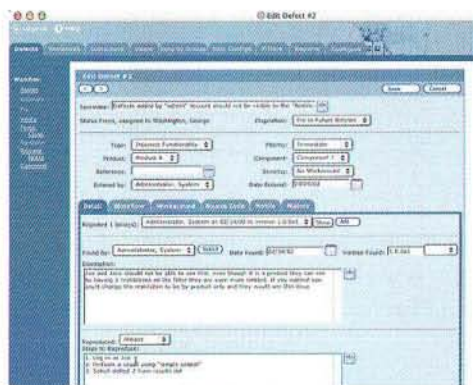


Figure 4. Defect Entry Screen

The displayed columns and sort order on list screens for each user are adjustable. The last settings saved are kept on the server, so that the user can see a consistent view when logging in from different machines or with different browsers. Users and Administrators are able to create filters to slice and dice the defect list any which way to include or exclude records based on attributes of any of the fields. Filters in use on a typical project would include: My assigned items, All Open Defects, High Priority Items, Items found in last day, and any other number of other combinations based on Status, Date, Product Line or other defect information.

The defect database can be queried with a quick search on summary and description fields or a complicated search using an advanced query builder. The query builder walks the user through selecting search criteria. Useful queries can be saved and turned into filters for later use by the individual, or to be shared with a group.

Server performance remains good across the WAN for groups working remotely from the server, and we have not experienced a performance drag despite having tens of thousands of entries in some databases that serve multiple product teams. The administrator has the choice of combining projects in one database or creating separate databases for each project. New databases can be built using previously customized projects as templates.

There is an import utility to move data from other systems into a new Testtrack Pro database. Comma and Tab Delimited text files can be imported directly. The import interface allows for the mapping of import columns to the destination columns in the program. The mappings can be preserved in an import template for future imports of text files from the same source application. There is a specified XML record format that allows for direct import and export of data from other Testtrack databases.

Editing a defect record places a lock on the record making it unable to be edited by other users. Locked out users can retrieve information to identify who has the record locked to avoid bottlenecks that may occur when a record is unintentionally left in edit mode. User sessions have an idle session timeout that is configured under the Server Options Section of the Server Admin tool.



THE LAW OFFICE OF  
BRADLEY M. SNIDERMAN

Need help safeguarding your software?

If you're developing software, you need your valuable work protected with trademark and copyright registration, as well as Non Disclosure Agreements.

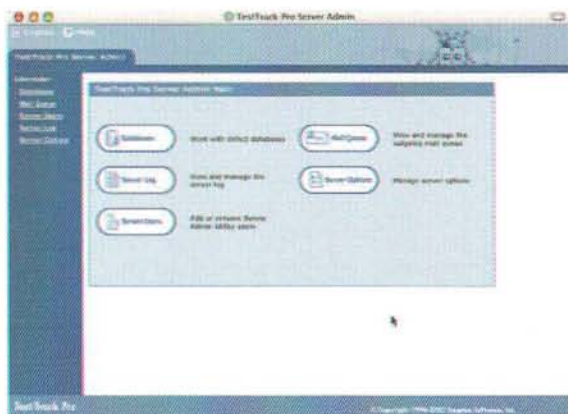
Then, when you are ready to sell it, you can protect yourself further with a licensing agreement.

I am an attorney practicing in Intellectual Property, Business Formations, Corporate, Commercial and Contract law.

Please give me a call or an e-mail. Reasonable fees.

23679 Calabasas Rd. #558 • Calabasas, CA 91302  
PHONE 818-222-0365 FAX 818-591-1038 EMAIL brad@sniderman.com





**Figure 5.** Server Admin Screen

### AUTOMATED NOTIFICATIONS

One useful feature of Testtrack Pro 6.1 is its automated E-mail notification system. During set-up the project administrator can identify the workflow actions that cause e-mail notification, and to whom. Notifications can be sent using Testtrack Pro's internal notification system or using SMTP or MAPI e-mail. The e-mail notification templates can be customized to contain pre-defined text as well as values from the defect record. Project managers may choose to receive notification for every new defect entered for their projects, while individual developers may only want notification when a task is assigned to them. Additional users can be added to the notification for a record for situations when a high priority issue comes to the attention of those higher up in the food chain who would normally not be included at that level.

### GET AT THE INFORMATION

Basic On-Demand reports are included with the Testtrack Pro installation. Reports can be customized using the report configuration menu, and they are formatted using included XSL (Extensible Stylesheet Language) files, which can be edited in the program itself, or replaced with your own XSL files. Three report types can be created by the user: detail, list, and trend reports. The trend reports are supposed to show actions plotted over time, such as defect rates reported during a specific period, but we have never found them to be much use, largely due it not being clear what criteria used in adding up the trend counts. The list reports are easy to set up and customize and are essential for defect review meetings and release planning. The same report generation engine is engaged to automatically build release notes based on a release level. Like the main reports, release notes are fully customizable as to their content and format.

### PUSH DEFECT ENTRY OUT TO THE CLIENTS

Seapine allows teams to extend the reach of the defect database out to their clients. Testtrack comes with a standalone defect entry application called Solobug that is extremely lightweight and can be sent out to customers, or anyone who is not connected to the server's network, but whose feedback on software issues is valued. Solobug runs on OS X, Palm OS, and Windows, and is customizable as to the fields it contains, the

values for those fields, and which fields are required for submission. A Solobug user saves the output file from Solobug after defect entry and e-mails it to a designated address so that it can be reviewed and imported into Testtrack if desired. The customization of Solobug includes provision for entering end-user instructions and directions on where to submit the Solobug output.

Solosubmit is an add-on to Testtrack Pro 6.1 that allows customers to enter defects and feature requests into the Testtrack database through a web page. As with Solobug and Testtrack, the fields, field names, field validation, and other items are easily customizable. Testtrack's workflow can be configured to segregate items entered through Solobug and Solosubmit, so that they are not entered into the development workflow until they have been reviewed internally. The intention of these tools is to lower support costs by allowing customers to enter items directly into your issue tracking system without having to telephone a support person and without having to re-key the information from an e-mail.

### TESTTRACK SDK WITH SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

If the team's workflow requires integration to a third party CRM tool (such as Siebel or SAP) or other in-house project management or customer service system, the Testtrack SDK allows access to Testtrack features and data through the SOAP programming interface. The SDK ships with platform independent SOAP client licenses, a Testtrack Pro SOAP Programming manual and a set of examples for such tasks as entering records programmatically. We used the SDK to build an integration between our Testtrack Pro databases and our CRM system, so that the Customer support team could enter records directly from the CRM system screen through a punch-out and automated log-in to testtrack.

### CONCLUSION

Testtrack Pro 6.1 is a comprehensive defect tracking solution that offers software development groups an affordable, scaleable, and reliable way to manage defects and enhancement requests throughout the software development life cycle. The ease of customization and low cost of server administration make it a tool that can fit the need of the smallest team, up through the largest, geographically distributed development group.

Seapine Software, the maker of Testtrack Pro 6.1, markets it as a "feature complete" defect tracking solution that can help reduce issue turn-around time, reduce time spent on project management tasks, and improve day-to-day communication for distributed teams. The automated notifications, comprehensive security settings, and On Demand reporting have helped our development and QA teams to stay informed about all issues regarding a release cycle, even when working from different offices and on different schedules.

The speedy performance of the Testtrack server allows us to host the databases in one location and allow access to Development and QA teams distributed across North America and Europe. It allows our engineers collaborate across distances and has become an integral part of our workflow.





# Peachpit

Essential books for the creative community

## Upgrading to Panther?

Let Peachpit help you make an easy transition to Apple's newest operating system. Our wide selection of titles caters to every learning level and style so you're sure to find just the book to tame your Mac's inner beast.

### Mac OS X Conversion Kit 9 to 10 Side-by-Side, Panther Edition

By Scott Kelby

0-7357-1389-8 • \$29.99

### Mac OS X Panther Killer Tips

By Scott Kelby

0-7357-1393-6 • \$29.99

### Mac OS X 10.3 Panther: Visual QuickStart Guide

By Maria Langer

0-321-21351-3 • \$24.99

### Mac OS X Panther Hands-On Training

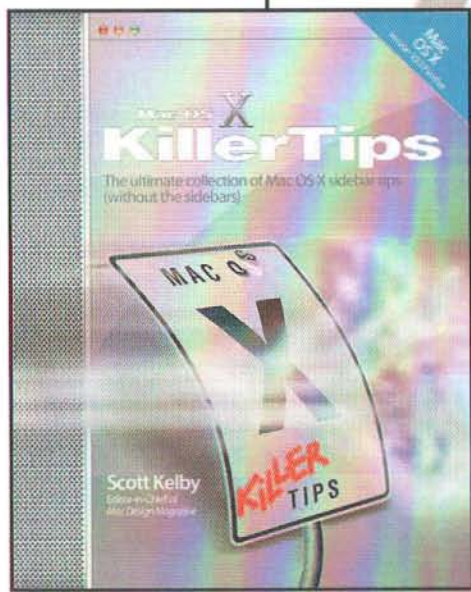
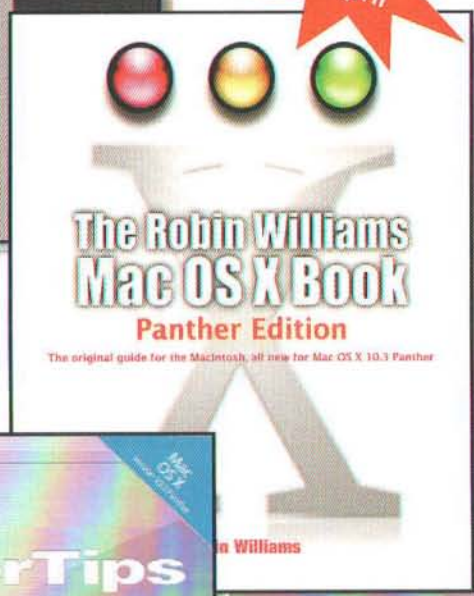
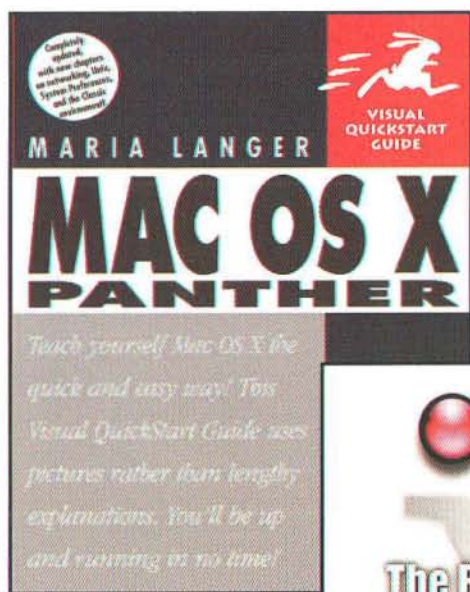
By Garrick Chow

0-321-24171-1 • \$35.00

### Mac OS X Security

By Bruce Potter, Preston Norvell and Brian Wotring

0-7357-1348-0 • \$39.99



## Coming in April!

### Robin Williams Mac OS X Book, Panther Edition

By Robin Williams

0-321-23296-8 • \$29.99

## Save 30%!



New  
Riders

Buy these books today at [www.peachpit.com/mactech0204](http://www.peachpit.com/mactech0204) and save 30% off their retail price, plus enjoy free domestic U.S. shipping. Just enter coupon code EM-F4AA-MTMF when you get to our checkout page. It's that easy!!



## List of Advertisers

Aladdin Knowledge Systems, Inc.....	5
Aladdin Systems, Inc.....	IBC
Ambrosia Software.....	11
Big Nerd Ranch, Inc.....	45
Brad Sniderman .....	69
DevDepot .....	67
DevDepot .....	28-29
Electric Butterfly .....	39
FairCom Corporation.....	1
Fetch Softworks.....	55
Full Spectrum Software, Inc.....	22
IDG World Expo .....	50-51
Lemke Software GmbH .....	4
MacDirectory .....	21
MacTech Magazine .....	61
Netopia, Inc.....	IFC
Paradigma Software .....	31
Peachpit Press .....	71
Pearson Education Communications .....	59
piDog Software.....	37
PrimeBase (SNAP Innovation) .....	47
Runtime Revolution Limited.....	BC
Seapine Software, Inc.....	33
Sophos, Inc.....	7
Utilities4Less.com.....	65
VVI.....	15
WIBU-SYSTEMS AG .....	13

## List of Products

Adobe Press * Peachpit Press .....	71
Big Nerd Ranch * Big Nerd Ranch, Inc. ....	45
c-tree Plus * FairCom Corporation .....	1
* DevDepot.....	67
Developers Library * Pearson Education Communications .....	59
Development & Testing * Full Spectrum Software, Inc.....	6822
Digital Rights Management * Aladdin Knowledge Systems, Inc. 5	
Fetch * Fetch Softworks .....	55
Graphic Converter * Lemke Software GmbH.....	4
HelpLogic * Electric Butterfly .....	39
IDG World Expo .....	50-51
InstallerMaker, Stuffit * Aladdin Systems, Inc. ....	IBC
Law Offices * Brad Sniderman .....	69
Long Distance Phone Service * Utilities4Less.com .....	65
MacDirectory * MacDirectory.....	21
MacTech Magazine Subscription * MacTech Magazine.....	61
Maximizing Your Mac! * DevDepot.....	28-29
piDog Utilities * piDog Software .....	37
PrimeBase * PrimeBase (SNAP Innovation) .....	47
Revolution 2.1 * Runtime Revolution Limited.....	BC
Software Protection * WIBU-SYSTEMS AG .....	13
Sophos Anti-Virus * Sophos, Inc. ....	7
Snapz Pro X 2.0 * Ambrosia Software .....	11
TestTrack Pro * Seapine Software, Inc.....	33
Timbuktu & netOctopus * Netopia, Inc.....	IFC
Valentina * Paradigma Software .....	31
Visual-Report Tool Developer * VVI.....	15

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.



# Multiple formats. Multiple platforms. Complex installers.

*Aladdin solves the compression and installation puzzle.*

**Trying to figure out how to handle multiple  
compression formats and platforms?**

*The StuffIt Engine solves the compression puzzle.*



Aladdin's StuffIt Engine SDK:

- Adds value to your application by integrating powerful compression and encryption.
- Is the only tool that supports the Stuffit file format.
- Provides a single API that supports over 20 compression and encoding formats common on Macintosh, Windows, and Unix.
- Makes self-extracting archives for either Macintosh or Windows.
- Available for Macintosh, Windows, Linux, or Solaris.

Licenses start as low  
as \$99/year

To learn more, visit:  
[www.stuffit.com/sdk/](http://www.stuffit.com/sdk/)

## **Stuffit Engine SDK™** The power of Stuffit in your software.



**Looking for the easiest and fastest  
way to build an installer?**

*StuffIt InstallerMaker completes your puzzle.*

It's not enough just to write solid code anymore. You still have to write an installer for your users. StuffIt InstallerMaker makes it simple and effective.

- StuffIt InstallerMaker gives you all the tools you need to install, uninstall, resource-compress or update your software in one complete, easy-to-use package.
- Add marketing muscle to your installers by customizing your electronic registration form to include surveys and special offers.
- Make demoware in minutes. Create Macintosh OS X and Macintosh Classic compatible installers with StuffIt InstallerMaker.

Prices start at \$250

To learn more, visit:  
[www.stuffit.com/  
installermaker/](http://www.stuffit.com/installermaker/)

## **Stuffit InstallerMaker™** The complete installation solution.™



[www.stuffit.com](http://www.stuffit.com)  
**(831) 761-6200**

© 2002 Aladdin Systems, Inc. Stuffit, Stuffit InstallerMaker, and Stuffit Engine SDK are trademarks of Aladdin Systems, Inc. The Aladdin logo is a registered trademark. All other products are trademarks or registered trademarks of their respective holders. All Rights Reserved.



# It'll Give You A Life.

O'ahu, Hawaii- Sun, sand and soft breezes make this one of the most relaxing vacation spots in the world. Don't forget the cool, fruity drink!

OAHU, HAWAII

PM  
15 JAN  
2004



Dear Revolution 2.1

You just keep on  
getting better.  
Wish you were here.

Missing you terribly,

Love,

Cecil

Revolution 2.1  
C/o Cecil's Computer

[www.runrev.com](http://www.runrev.com)

24-7 Relief From Aggravation  
(formerly DullAndDreary Coders)  
The Corporate World, #1-EASY

# But A Geek Is Always A Geek.

**Revolution 2.1: the English like language  
designed around the way you think.**

Develop and deliver on 14 platforms, including Mac OS X, Windows and Linux.  
Now with support for XML, additional SQL databases, video capture, Unicode,  
Reports, enhanced faceless CGI applications, and more.

And now from Dan Shafer, the first "how to" book on Revolution.  
Get a headstart with "Revolution: Software at the Speed of Thought",  
volume one now shipping from [www.runrev.com/revpress](http://www.runrev.com/revpress).  
Thousands of developers have already joined the Revolution. Can you afford to wait?  
Pricing starts at \$149. Don't let the revolution in coding start without you.

**User Centric Development Tools**



Runtime Revolution • 91 Hanover Street • Edinburgh EH2 1DJ • UK  
Phone +44 (0)131 718 4333 • Fax +44 (0) 845 4588487 • [www.runrev.com](http://www.runrev.com) • Email [info@runrev.com](mailto:info@runrev.com)

**Revolution  
Studio**



**winner  
macworld  
eddys**